2.15 SITUATION UPDATE

The purpose of the Situation Update Functional Element (FE) is to update the pilot's knowledge of the current tactical situation. The update process consists of placing information about the pilot's own aircraft and other observed entities (aircraft and missiles) into the pilot's situation awareness, or "mental model", and then evaluating relationships between the pilot and observed entities. The situation update can be broken down into two parts, making observations and processing those observations into the pilot's mental model.

Pilot knowledge of his own aircraft comes from observations of his cockpit instruments, HUD, and through direct observation through the cockpit canopy. Knowledge of other entities comes to the pilot from his sensors, by direct visual observation, from radio messages from friendly aircraft, or from inferred detections.

Sensors available to the pilot include: visual (eyes), radar warning receiver (RWR), identify friend-or-foe (IFF), infrared search and track (IRST), radar, missile approach warning radar (MAW) and missile launch warning (MW) devices. The pilot can only gain information from a sensor if he actually looks at the display for that particular sensor. For example, if the pilot spends his available search time looking out the canopy, he will not have time to look at the radar screen and pick up any target information displayed there. A part of the pilot's duty cycle includes time allotted to make new observations. During this time, the pilot will attempt to look at the sensors that have the highest priority at that time. Sensor search priorities are based on intrinsic sensor value, perception of what information is being displayed on that sensor, and the time since that sensor was last observed. Sensors will be searched by the pilot starting with the highest priority sensor and continuing down the sensor priority list until either all sensors have been searched or the search time has expired.

Pilots may also receive radio messages from friendly sources. This may include information on hostile detections, missiles fired by the friendly, missiles detected, or command and control information from GCI or AWACS controllers.

In addition to the above, a pilot's situation awareness may be changed through one more mechanism, referred to as inferred detection. An example of an inferred detection would be a situation where a pilot observes a wing man destroyed by a missile, but he has not previously observed any hostiles. In such a case, he will infer the presence of hostile aircraft.

Once the observations of other aircraft and missiles have been made, then the information must be assimilated into the pilot's mental model. Determinations will be made whether the detected aircraft or missiles are new or were previously detected, if the detected aircraft is hostile, friendly, or unknown, if an aircraft has been killed, or what has happened to known aircraft that were not observed in this period. Other information such as intent to fire messages from other friendlies and orders from the flight leader are also folded into the pilot's mental model.

The situation assessment process begins with the incorporation of new observations into the mental model of a pilot. The object of this initial phase is to determine a physical state (e.g., position, velocity) of each aircraft and missile, consistent with both the new observations and with the previous assessment of the situation. This process is called tracking. The input to the tracking model consists of successive observations of each target.

The output of the tracker (at a particular time) is the estimated state vector of each target considered and an estimate of the errors associated with the state vector.

2.15.1 Functional Element Design Requirements

This section presents requirements necessary to implement the Situation Update FE. This function will simulate changes to the situational awareness of a pilot via a process of making observations, interpreting their significance, and storing information in arrays that can be accessed during the decision making process. The function will be executed repeatedly for each pilot in the simulation so that a model of current situation awareness can be maintained over the course of simulation execution.

a. Brawler will simulate pilot observations of his own aircraft. These will include both performance and physical state parameters, which will be used to update portions of a data model of the pilot's current perception of the situation.

b. Brawler will simulate pilot observations of other aircraft and weapon system platforms. These will include information on position, status, and perceived hostility of those platforms, which will be used to update portions of a data model of the pilot's current perception of the situation.

c. Brawler will simulate visual observations of cockpit gauges, indicators, and sensor displays as well as visual observations of objects and weather outside the cockpit. These observation scans may be ordered according to priorities established during previous observation periods that result in perceived importance of certain sensors or visual sectors.

d. Brawler will simulate receipt of communications via radio and sensors that display information derived via communications with other sensors.

e. Brawler will process observations to determine their status and correlation with previous observations and will assign factors that may affect subsequent observations and/or influence pilot decisions to act.

These requirements will be satisfied by the combined implementation of the design elements described in the following section. They were inferred from knowledge of how the existing models currently perform the Situation Update function.

2.15.2 Functional Element Design Approach

This section contains a description of the design approach used to implement the design requirements outlined in the previous section.

The pilot model situation update can be separated into two parts, updating the pilot's knowledge of his own aircraft and updating the pilot's knowledge of other entities. In the real world, information about his own aircraft is always available to the pilot via cockpit instrumentation and his HUD. The approach taken to simulating pilot awareness of this information is to perform a straightforward transfer of aircraft status information from the physical system model of the aircraft to the pilot's mental model.

Observation of other platforms can occur through radio messages received, inferred detections and sensor observations, counting direct visual observation as a kind of sensor observation. Only those entities that are observed by the pilot in these ways are allowed into the mental model. An exception is made to this during the first consciousness event, where knowledge of unseen entities may be allowed into the mental model as a simulated observation history, e.g., flight mates may be visually observed even if not currently visible to the pilot due to visual obstruction by the pilot's airframe, since the pilot can be reasonably assumed to know who is in the flight and where they are when the simulated engagement begins.

When making sensor observations, the pilot must first determine which sensors to look at. The sensors are ranked in order of priority. Each sensor is given an intrinsic value, based on the relative usefulness of the information it nominally provides. This value is then adjusted to reflect the time elapsed since the sensor was last observed. Each sensor's value is further adjusted based on aircraft and missiles that the pilot believes to be in the field of view of that sensor. Once the sensors have been ranked, the pilot searches the highest ranking sensors as long as time permits. The observations from these sensors are used to update the pilot's situation awareness.

To keep this description manageable, analogous functions will be noted and not described in full detail. For example, the code that adjusts a sensor's priority based on a perception that missiles are in the field of view of the sensor is very similar to the code that adjusts a sensor's priority based on a perception that aircraft are in the field of view, so this design will only fully address adjustments due to aircraft. Similarly, sensor priority adjustments due to aircraft have similar code for each type of sensor display and this discussion will be limited to observation of the radar display.

Once observations have been made, they must be correlated with the information already present in the pilot's mental model. A correlation must be made between each observed aircraft and the aircraft already known to the pilot to determine if this is a new or previously detected aircraft. If it is a new aircraft, it is added to the pilot's mental model. If the aircraft had been previously detected, its track in the pilot's mind is updated if the aircraft is perceived to still be alive. If it is observed to be dead, then it is added to a list of known dead aircraft and is otherwise removed from the pilot's mental model. The next determination is whether the observed aircraft is friendly, hostile or unknown. Finally, a projection is made for aircraft that are known to exist but were not observed in this consciousness event. Processing missile observations is similar to processing aircraft observations, radio observations, and inferred detections.

The pilot model situation update is performed by the procedure shown in Figure 2.15-1. Each numbered entry is a design element. A description of each design element is given in the paragraphs that make up the remainder of this section. Italic text denotes elements that are not directly related to the observation of aircraft on the radar scope and will be left for future documentation.

Update knowledge of ownship (1.0)
- Update ownship performance parameters (1.1)
- Update ownship physical state (1.2)

Update knowledge of other platforms (2.0)
- Make observations of other platforms (2.1)
  - Sensor observations (2.1.1)
    - Determine which sensors to look at first (2.1.1.1)
      - Assign intrinsic priority values to each sector (2.1.1.1.1)
      - Adjust priority for aircraft perceived being in that sector (2.1.1.1.2)
        - Radar sector (2.1.1.1.2.1)
        - *Visual sector*
        - *RWR sector*
        - *IFF sector*
        - *IRST sector*
        - *Integrated display sector*
      - *Adjust priority based on missiles perceived being in that sector*
        - *MAW sector*
        - *RWR sector*
        - *MW sector*
        - *IRST sector*
        - *Integrated display sector*
        - *Radar sector*
        - *Visual sector*
    - Search the highest valued sectors while time permits (2.1.1.2)
      - Radar search sector (2.1.1.2.1)
        - Observe radar scope (2.1.1.2.1.1)
      - *Visual search sector*
      - *IRST search sector*
      - *IFF search sector*
      - *MW search sector*
      - *MAW search sector*
      - *RWR search sector*
      - *Integrated display search sector*
    - Make observations (2.1.1.3)
      - Radar observations (2.1.1.3.1)
      - *Visual observations*
      - *IRST observations*
      - *IFF observations*
      - *MW observations*
      - *MAW observations*

FIGURE 2.15-1. Situation Update Procedure (Page 1 of 2).

RWR observations
Integrated display observations
Radio messages received
Inferred detections
Process observations of other platforms (2.2)
Process aircraft observations (2.2.1)
Determine if aircraft is new or previously detected (2.2.1.1)
Incorporate observations of previously undetected aircraft (2.2.1.2)
Incorporate observations of previously detected aircraft (2.2.1.3)
Determine if aircraft is friendly, hostile or unknown (2.2.1.4)
Estimate known aircraft that were not observed this cycle (2.2.1.5)
Process missile observations
Determine if missile is new or previously detected
Incorporate observations of previously undetected missiles
Incorporate observations of previously detected missiles
Determine the missile's owner if possible
Determine the missile's target if possible
Estimate known missiles that were not observed this cycle
If missiles were observed, rethink pilot posture
Process radio observations
Detected aircraft messages
Detected missiles messages
Intent to fire messages

FIGURE 2.15-1. Situation Update Procedure (Page 2 of 2).

Design Element 15-1: Update Knowledge of Ownship

Pilot knowledge of his own aircraft is updated in two parts, which are detailed below. The first part, implemented under subroutine *perfrm*, updates the pilot's knowledge of his aircraft's current performance capabilities, and the second part, implemented under subroutine *ccself*, updates his knowledge of his aircraft's current physical state.

Design Element 15-2: Update Ownship Performance Parameters

Under this design element, subroutine *perfrm* updates the pilot's knowledge of his aircraft's current performance capabilities. This is done by updating the values of a number of factors that describe the aircraft's performance capabilities. These include:

maximum sustained Gs
corner velocity
drag versus lift tables
angle of attack versus lift tables
maximum turn rate
drag at maximum sustained Gs
effective gross thrust
effective propulsive drag

current drag on aircraft
minimum longitudinal acceleration
maximum longitudinal acceleration
minimum allowable angle of attack
maximum allowable angle of attack
drag versus angle of attack tables
longitudinal force versus throttle setting

The computation of all variables except for maximum sustained Gs and corner velocity is straightforward and is based upon the current ground truth state of the aircraft. The maximum sustained Gs is intended not as the true maximum sustained G capability, but rather as the maximum Gs that the pilot will use in any situation except extreme emergency. It is sensitive to a number of non-aerodynamic factors, including interest in firing a weapon and range to the nearest hostile or hostile missile.

Design Element 15-3: Update Ownship Physical State

Under this element, subroutine *ccself* updates the pilot's knowledge of this aircraft's physical state. Some of the ownship status information transferred to the pilot's mind includes: inertial position, velocity, and acceleration, mach number, angle of attack, speed, aircraft mass and dynamic pressure. Ground truth information is used as the source of this information.

Design Element 15-4: Update Knowledge of Other Platforms

Pilot knowledge of other entities is updated in two steps. The first step determines what observations the pilot will make and what information is contained in those observations. The second step incorporates the observations into the pilot's situation awareness. Subroutine *conevt* acts as the executive routine for updating the conscious pilot's knowledge of other platforms. The two primary subroutines called from *conevt* to update the conscious pilot's knowledge of other platforms are *sensor*, which is the executive routine for making sensor observations and *mindup*, which is the executive routine for processing observations of other platforms into the conscious pilot's mental model.

Design Element 15-5: Make Observations of Other Platforms

Observations can be constructed from received radio messages, visual observation of sensor displays, or from inferred detections. Sensor observations, including direct visual observations, are handled by subroutine *sensor*. The functions related to sensor observations are detailed below.

The observations made of each target are generated by the sensor models of visual, radar, communicated, and avionics display observations. The form of an observation differs for different types of sensors and sensor modes. For example, when a pilot makes a visual sighting, he observes range, azimuth, elevation, and, with lower accuracy, aspect angle, azimuth rate, and elevation rate. For a radar in scan mode, range, azimuth, elevation, and range rate are observed, but not aspect angle.

There is also a cross correlation matrix associated with observations. It represents the distribution of errors in the measurement with respect to the true target state. At present the

sigmas of the observables are squared and inserted into the corresponding diagonal elements of the observation cross correlation matrix.

Design Element 15-6: Sensor Observations

In order to make observations, a pilot must visually observe an area of the sky or an avionics display. In Brawler, these different sources of visual observations are called search sectors. There are eight sky sectors and one sector for each sensor display. If the aircraft is equipped with a sensor fusion device, then all of the separate avionics displays are replaced with a single integrated display (IND). The observation process is modeled by computing the relative values of each available search sector, and then observing each one, from most important to least, until the time available for making observations has been used up. Subroutine *sensor* is the executive routine for conducting sensor searches.

The design elements that follow trace through the sensor observation process along the path for the radar search sector. The other search sectors (visual, IFF, IRST, etc.) have analogous processing paths and will not be discussed in detail. Similarly, observations of aircraft and missiles are handled in similar ways, so detailed description is given in this report only for aircraft observations.

An avionics display is also a generator of observations. It is assumed that the track bank of the device is displayed somewhere on the instrument panel. This display can include some or all of the variables in the track bank. The pilot perceives the observation with errors that depend on the resolution of the display output. In this way it is possible to model the independent device behavior and the pilot's perception of the device output separately.

Design Element 15-7: Determine Which Sensors to Look at First

The search sectors are ranked so that the pilot can choose which sensors to look at first. Each search sector is scored on three factors, intrinsic value, time since last observed and whether or not the pilot believes that sensor will contain any aircraft or missile information. The highest scoring search sectors will be looked at first during the pilot’s available search time.

Design Element 15-8: Assign Intrinsic Priority Values to Each Sector

Sector search values are initialized with user input parameters for each sector that indicate their relative importance. These values are then adjusted to reflect the time elapsed since the search of each sector. There are currently 18 search sectors, listed as follows:

1-8) visual search sectors, each 1/8th of a sphere centered at the pilot
9) radar
10) infrared search and track (IRST)
11) identify friend or foe (IFF)
12) missile warning (MW)
13) radar warning receiver (RWR)
14) missile approach warning (MAW)
15) integrated display (IND, i.e. sensor fusion device (SFD))

16) reserved for proprietary use
17) reserved for proprietary use
18) reserved for proprietary use

Avionics sectors are skipped if the aircraft does not carry an avionics device of that type.

The adjustment factor for time since the sector was last searched is:

$$1 - \text{Cauchy}(T_{now} - T_{sch}, agesec)$$

where T_{now} is the current simulation time, T_{sch} is the time of the last search of this sector, and *agesec* is an input data element specifying the desired revisit interval for this sector. *agesec* is used to set the width of the Cauchy function. For visual sectors, a random value ranging from 0 to .001 is also added to break ties between visual sector scores.

Design Element 15-9: Adjust Priority Based on Aircraft Perceived to be in That Sector

Search sector priorities are adjusted by counting and identifying aircraft already known to the pilot that might appear in each observation sector and then increasing the score of the sector search values accordingly. The purpose of this is to simulate the fact that a pilot will tend to revisit a sensor display or sector of the sky more frequently if he already knows that it contains a lot of potentially important information (i.e., is displaying a lot of tracks). The algorithm used in this design element is to loop through the list of aircraft known to the pilot, and for each one to determine all search sectors that might contain the aircraft. This determination is based upon observer target geometry and FOR coverage of each sensor. The target aircraft is added to a list of potential targets for each sector in which it might be observed and the sector search value for that sector is updated. Note that there is only one IFF sector, even if the aircraft is carrying multiple IFF devices. The IFF sector value will be that of only the most valuable IFF device.

Design Element 15-10: Radar Sector - Adjustment due to Aircraft

This design element, implemented by subroutine *rdrval*, calculates the incremental value to be added to the radar sector search value as a result of aircraft being in the radar sector (observable by the radar). Different weights are attached to the value depending on whether the target is believed to be a friend or foe, whether the target is the primary target or if the radar is trying to lock on to the target in Single Target Track (STT) mode. The incremental search value is added to the current sector search value.

Design Element 15-11: Search the Highest Valued Sectors While Time Permits

Once the sector values have been assigned, the sectors are searched in order, from highest to lowest priority using an iterative algorithm. During each pass through the loop, the highest valued sector is determined, then the appropriate search algorithm for that sector is executed to determine if any observations are made and how much time was spent searching the sector. The sector search value for this sector is then set to zero so that on the next pass through the next highest scoring sector will be processed. This continues until the search time is exhausted, or until there are no sectors remaining to be searched.

It should be noted that the value of searching a sector depends upon what entities the pilot believes to be in that sector, but the observations made are based upon the actual contents of an avionics display or, for visual observations, the positions and orientations of the actual aircraft and missiles.

Design Element 15-12: Radar Sector Search

Observation of the radar display is broken into two parts, discussed below. The first part, observing the radar scope, examines the current status of the radar model to determine what is displayed on the radar scope. The second part, making the observations, takes this information and translates it into zero or more observations.

Design Element 15-13: Observe Radar Scope

The purpose of this design element is to construct a list of the aircraft and missiles that the pilot observes when he looks at his radar scope. Each radar track is tested to see if it will be observed and, if so, is added to a list of entities that will be observed. A track will not be observed if:

f. It has not been updated since the radar scope was last looked at.

g. It is beyond the radar scope range limit.

h. It has faded from the scope, due to maximum display time limits. A radar in scan or STT mode will display each hit on a target for a finite amount of time, then it fades out. The amount of time is an input data item.

i. It is a TWS track, but it is not established. A radar in TWS mode must make *m* detections of a target in every *n* frames (detection opportunities) in order to mark a track as "established." Established tracks are displayed to the pilot and are available for use by the fire control device. The values of *m* and *n* are input data items.

Design Element 15-14: Make Observations

As each sector is searched by the pilot, a list of observed entities is constructed. Each is translated into an observation, whose set of observed variables is a function of the type of observation made. For example, some sensors may only observe target azimuth and elevation, others may observe additional variables such as range, range rate, or the wavelength of an RF emission.

Design Element 15-15: Radar Sector Observation List

At this point, each entry on the list of observed entities is translated into an observation. The set of observables associated with each observation is a function of radar mode, submode, and whether or not the detection has been jammed. If the radar is in TWS or STT mode, the observables are target position and velocity in earth centered Cartesian coordinates. If the radar is in scan mode, then the observables will be azimuth and elevation if the detection is being jammed, but not strongly enough to completely mask the target. The observables will be azimuth, elevation, and rage rate if the waveform used does not allow range measurement (specified in input data) or if the radar is in high PRF and in the

velocity search submode. In all other cases, the observables will be range, azimuth, elevation, and range rate. The observables will not be perfect "ground truth" values, but will reflect the errors and uncertainties of the radar track or detection from which they are derived.

For each observed entity, the observables are computed and recorded, along with detection method and any aircraft typing information. The detection method is either a clear radar detection or a jammed detection. In most cases, radar detections do not carry with them any typing, or ID, information about their targets. However, there is an operating mode switch that will allow typing of detected targets that are trying to jam the radar.

Design Element 15-16: Process Observations of Other Platforms

The second major part of the situation update involves processing the observations into the pilot's mental model. Subroutine *mindup* is an executive routine for processing entity observations. Information obtained through radio messages is processed first. Communicated sightings of other aircraft and missiles are added to the pilot's mental model first. Next, intent to shoot messages are processed, which may cause the pilot to change targets if the target he has selected is already being attacked by another friendly. Sensor observations are processed next. If an observation is determined to be a new aircraft sighting, it is added to the pilot's mental model. Otherwise the existing mental model track of the observed aircraft is updated to include the new observation information.

As part of this update, assessments are made to attempt to determine aircraft side (hostile/friendly/unknown) and missile ownership and target. If the pilot's situation has changed considerably due to the new information contained in his latest observations, he must make new decisions relating to the flight posture or flight tactics that he will employ. Factors that may require the pilot rethink how to address his new situation include the observation of a new aircraft or missile, the identification of a previously unknown aircraft to now be hostile or friendly, and the receipt of a change in his orders. Finally, outgoing messages are prepared to be sent to flight mates. These messages contain information from the latest observations and responses to any previously requested data.

Design Element 15-17: Process Aircraft Observations

Observation information for each detected aircraft is compared to the pilot's knowledge of each aircraft already known to him to determine if the target is a new aircraft or a previously known aircraft. A new aircraft's relationship to the conscious pilot is assessed, if possible, based upon the observables of the observation. If the detected aircraft is already known to the pilot, the relationship between himself and the observed aircraft is updated if the latest observation contains new information about aircraft type, side, or alive/dead state. The establishment and update of aircraft relationships are covered in the remaining design elements.

Design Element 15-18: Determine if Aircraft is new or Previously Detected

After an attempt is made to determine if the detected aircraft is already known to the pilot, a decision must be made whether to accept or discard the observation. If the observation is accepted and a correlation exists between the observed aircraft and one of the aircraft known to the pilot, the match is noted and the observation information is merged with the

aircraft's track in the pilot's mind. If no correlation exists, the aircraft is marked as new and a track is initialized for it in the pilot's mind.

An observation may be discarded if another observation made in the same consciousness event provides better information or if the observation contains no new information. A detection is discarded for the following cases: 1) discard an electronic countermeasure (ECM) or IRST detection if it is accompanied by a visual observation, 2) discard an IRST detection if it is accompanied by a radar observation and, 3) discard an ECM message detection if the detected aircraft is already known to the pilot. An ECM message detection occurs when a message sender is informing a message receiver about a target for which he has only ECM information.

Design Element 15-19: Incorporate Observations of Previously Unknown Aircraft

If an aircraft is detected for the first time, but is already dead, then the observation is discarded and the aircraft is ignored. Otherwise, newly observed aircraft need to be initialized in the pilot's mental model. First, a "track" recording the state vector (position, velocity, acceleration, orientation) of the aircraft is initialized. The new aircraft's state vector is initialized in the pilot's mental model using either perfect or imperfect information, depending upon the operating mode specified by the analyst. When using perfect information, the new aircraft's state vector is taken from ground truth, projected to the current simulation time. For imperfect information, a track is initialized with nominal values, then updated using the observables contained in the observation. The update is done via a Kalman filter. In addition to creating the track, an attempt is made to determine the pilot's relationship to the new aircraft. The relationship includes hostile/friendly status and, if friendly, whether the aircraft is a flight or element mate. Finally, if the observation contains typing information, this is also transferred into the pilot's mental model.

Design Element 15-20: Incorporate Observations of Previously Known Aircraft

The process for updating the pilot's perception of a known aircraft with a new observation is similar to that for an initial detection, except that an existing track is updated instead of creating a new track. Relationship and typing information are also reassessed if the new observation contains additional information.

Typing and relationship information will only improve as a result of new observations, they will never be degraded based upon new information. Brawler does not play imperfect correlation or incorrect typing, so pilots will either make correct assessments or none at all. They will never misidentify other aircraft.

When a pilot has determined that a previously known aircraft has been killed, that aircraft is added to a list of aircraft known to be dead and is otherwise removed from his mental model. The pilot may determine the aircraft was killed either by visual observation, through a received radio message. He may also assume that a flightmate is dead if no radio responses have been received from the flightmate in a sufficiently long time (between ten and thirty seconds from the last query to that aircraft, depending upon how much radio jamming is present), or if he is able to continue to track it for more than five seconds after it is killed.

Design Element 15-21: Determine if Aircraft is Friendly, Hostile or Unknown

After all of the new observations have been processed, the pilot will attempt to infer the types of any aircraft whose types are unknown to him. Inferred identification is based upon association with other aircraft whose types are known. Each other aircraft in the unknown's flight is checked, and if the pilot knows the type of the other aircraft and if the unknown is flying in formation with that other aircraft, the pilot is allowed to infer the type of the unknown. Flying in formation is defined in terms of a formation factor:

$$frmfac = dx/rnglim + dv/vlim$$

where *dx* is the separation, *dv* is the magnitude of the difference in velocities, *rnglim* is one nautical mile, and *vlim* is 25 feet/sec.

If the formation factor is less than or equal to 2.0, the two aircraft are considered to be in formation.

Design Element 15-22: Estimate Known Aircraft That Were not Observed this Cycle

Pilots must account for aircraft that are known, but were not observed during this consciousness event. Brawler uses several algorithms to update pilot perception of the state vector of another aircraft in this situation, depending upon geometry and user specified model options. These are: perform a perfect information update, extrapolate the target position using a Kalman filter, perform a constant velocity projection, employ the pseudo-observation technique, or assume the aircraft is dead and remove it from the pilot's mental model. If the model is running in the perfect information mode, then ground truth information is used to update the pilot's knowledge of the target.

The pseudo-observation technique updates the track of the unobserved aircraft by creating data that will be used to update the track just like any other observation, but with large uncertainties associated with it. The reason for doing this is that in real life, an experienced pilot can often guess the actions of another pilot he cannot observe. By allowing very limited, error rich access to the actual location of unseen targets, this guessing process is modeled well enough to suit most engagement scenarios. This methodology has in mind a within visual range blowby, with the pilot guessing which way his opponent will break. Pseudo-observation updates are only performed if the target is within 5 nmi of the pilot and if the pilot's knowledge of the position of the target is sufficiently accurate. Sufficiently accurate is defined as having an uncertainty in direction corresponding to 10 degrees or less at a distance of 5 nmi. Pseudo-observations can also be turned off by the user by setting an operating mode switch in the **SCNRIO** file.

If enough time has elapsed since the aircraft's last observation (currently 300 seconds) the unobserved aircraft is assumed to be dead or to have left the engagement, so it is removed from the pilot's mental model. If pseudo-observations are not used, and the aircraft is not removed from the mental model, then the aircraft is projected forward using one of two algorithms, depending upon the importance of the aircraft to the pilot. If the aircraft is one of the aircraft that the pilot is actively considering when making decisions (a member of the so-called "detailed consideration group"), then a Kalman filter update is performed, with exponentially decreasing acceleration (time constant of 5 seconds) and vertical velocity

(time constant of 20 seconds) based upon the time since the last observation. If the aircraft is not in the detailed consideration group, its perceived acceleration is set to zero and a constant velocity projection is made. If this takes the target to an altitude below 1000 feet, then the altitude is set to 1000 feet.

In addition to updating the pilot's perception of the location and velocity of unobserved aircraft, the pilot may also lose confidence in his identification of an unobserved aircraft. This will happen if an aircraft whose type is known is close enough to another aircraft that is either unknown or on the opposite side, and the pilot fails to observe both of them for a long enough period so that he can no longer tell which is which without new identifying information. A range limit is computed based upon the time since the aircraft was last observed. The range limit is zero until 20 seconds have elapsed, then increases linearly to a maximum of 4 nmi at 40 seconds. If any unknowns or aircraft on the opposite side are within this limit of the unobserved aircraft, then the pilot loses confidence in its ID and typing information is lost.

Tracking

Once it has been determined that an observation is of a particular target, the new information must be merged with previous information about that target which has been derived from previous observations. This process is the heart of the tracker, which uses a modified Kalman filter. The filter folds observations into the current opinion of the state for each target and derives a new opinion of the target's state. The algorithm is detailed in Appendix I of the Brawler Analyst Manual.

Correlate

Part of the tracking process is the determination of whether an observed target corresponds to one already in the pilot's mental model. The Brawler model of this process is currently rather limited. For visual, communicated, and avionics display observations, perfect correlation of a known target with a new observation of that target is assumed. The model is a bit more sophisticated for the radar case, in that confusion can take place regarding targets in the same radar resolution cell or within the coverage of an ECM strobe.

Update

When an avionics device with an internal track bank makes an observation, the filtering process is performed on the basis of a best fit to a constant velocity target model. For all other cases the observation process uses a constant acceleration model.

A mode is available in Brawler, historically called "perfect information mode" (although "perfect tracking" is a more appropriate term), in which the entire tracking process described above is short circuited, except for the case when observations are made under ECM conditions. When this short circuiting occurs, pilots are given exact knowledge of each known target's state. It is important to note that this applies only to known aircraft and missiles; initial detections must still take place by legitimate means. The perfect tracking mode is intended as a debugging mode, since its use allows one to criticize pilot decisions without worrying about whether they are being influenced by their limited ability to correctly assess the position and velocity of other targets. A partial exception is made in the case of some avionics devices, since, if they are active, one presumably ***wants*** to observe their confusing effects.

2.15.3 Functional Element Software Design

This section contains the software design necessary to implement the functional element requirements and the design approach defined in the preceding sections. The first subsection describes the subroutine hierarchy and describes how the subroutines work to make up the situation update. The remaining subsections contain functional flow diagrams for the main subroutines and describe all important operations represented by each block in the diagrams.

Situation Update Subroutine Hierarchy and Description

The major routines comprising the situation update algorithm and their purpose are given below with the indentation of the routine name used to indicate the level of the routine within the calling tree.

conevt- pilot consciousness event executive

- *perfrm* - computes performance variables for the pilot's aircraft
- *sensor* - executive routine for conducting sensor searches
  - *setsvl* - initializes sector search priority
  - *acnear* - modifies sector search priority if aircraft are perceived by the pilot to be in that sector
  - *sctsch* - makes sector search decisions and conducts sector searches
- *mindup* - executive routine for updating the mental model of the conscious pilot
  - *ccself* - updates pilot knowledge of his aircraft's physical state
  - *cc2x0* - merges aircraft detections with the existing information about aircraft in the pilot's mental model
    - *cc2x1* - correlates observations with targets in the mental model
    - *mremac* - deletes aircraft from the current mental model
    - *cc2x2* - updates perception of already known aircraft with new observations
    - *cc2x3*- adds newly detected aircraft to the mental model
    - *cc2x4*- updates mental model tracks of aircraft that are already known, but that were not observed this time

A number of secondary and utility subroutines are also used in the situation update process. These are listed below. This list contains a large number of routines that are outside the scope of this FE, but are included here as an aid in reading and understanding the functional flow diagrams and the code.

add_2_sensed	Makes consistency checks and adds observations to the /*sensed*/ common blocks.
asklev	Requests decisions to be made at different levels based on input.
asown	Assesses missile ownership, if possible.

asstgt	Gets target assigned to pilot by flight leader.
astgt	Assesses missile target and threat to self.
astyp	Assesses the type of the observed aircraft based on knowledge of type of other members in its flight.
bstsec	Determines which sector has the highest priority and should be searched next.
cc2_gcirng	Checks for range data from GCI.
ffsrch	Makes an observation of a particular IFF device.
flyac	Updates the state vector for the current aircraft.
fncom	Sends messages prepared for flight mates.
fncomi	Initializes message sending.
gcetim	Determines the time interval to the next self called consciousness event.
getsec	Calculates which sectors the aircraft under consideration resides in.
grdrc	Gets radar characteristics data for the conscious pilot's aircraft.
grdrs	Gets radar status data for the conscious pilot's aircraft.
iffsch	Performs search of the IFF sector and determines if any detections are made.
iffval	Calculates the incremental value to be added to the IFF sector search value as a result of aircraft being in the IFF sector.
incsec	Updates the array containing the number of aircraft/missiles in the visual sector.
indobs	Constructs observations based upon the contents of the integrated trackbank of the sensor fusion device.
indsch	Performs search of the integrated display sector and determines if any detections are made.
indval	Calculates the incremental value to be added to the integrated display sector search value as a result of aircraft/missiles being in the integrated display sector.
inferl	Makes inferred observations.
inicmm	Sets up radio message receipt for the conscious pilot.
irsobs	Performs the actual sector search for the IRST sector.
irssch	Performs search of the IRST sector and determines if any detections are made.

irsval	Calculates the incremental value to be added to the IRST sector search value as a result of aircraft being in the IRST sector.
irsvlm	Calculates the incremental value to be added to the IRST sector search value as a result of missiles being in the IRST sector.
kalmni	Propagates an aircraft track using a Kalman filter.
m3perf	Places perfect information on the tracking arrays if the perfect information flag is set.
majud	Does a complete situation assessment update.
makece	Plants the next consciousness event for the pilot into the event list.
mawobs	Uses the contents of the MAW device trackbank to construct missile observations.
mawsch	Performs search of the MAW sector and determines if any detections are made.
mawval	Calculates the incremental value to be added to the MAW sector search value as a result of missiles being in the MAW sector.
mindin	Retrieves internal memory for the current pilot's aircraft.
minud	Performs a partial situation assessment.
mmordr	Reorganizes a pilot's mental model if necessary.
modsel	Selects an action based on the updated mental model.
msl2x0	Merges missile detections with the existing information about missiles in the pilot's mental model.
msl2x1	Correlates detections with already known missiles.
msl2x2	Tracks and updates known missiles in the presence of new observations of those missiles.
msl2x3	Initializes new missiles from their observations.
msl2x4	Takes care of known missiles unobserved in this consciousness event.
msl2x5	Checks for active radar missiles that have acquired their target.
msl2mm	Adds new missiles to mental model and increments the number of observed missiles.
mslner	Adjusts the sector search priority values if missiles are perceived to be present in that sector.
mslsec	Calculates which sectors the missile under consideration resides in.

mslpob	Makes aircraft track projections using the pseudo observation technique.
msvpob	Makes aircraft track projections using the pseudo observation technique.
mwmsl	Allows the pilot to observe each missile in the MW track bank.
mwsch	Performs search of the MW sector and determines if any detections are made.
mwval	Calculates the incremental value to be added to the MW sector search value as a result of missiles being in the MW sector.
obrdsc	Provides a list of aircraft and missiles that the pilot will be allowed to observe when he looks at his radar scope.
pcode	Exercises user defined code (production rules).
premob	Prepares observational messages about missiles.
preobs	Prepares observational messages about aircraft.
prereq	Prepares request messages for unobserved aircraft that the pilot is sufficiently concerned about.
radar	Executive routine for making radar screen observations.
radobs	Computes radar errors and cross correlation matrices.
radvlm	Calculates the incremental value to be added to the radar sector search value as a result of missiles being in the radar sector.
rcv_intent	Processes intent to shoot messages.
rcv_orders	Processes messages regarding new orders.
rcvmob	Receives radio messages about missiles.
rcvobs	Receives radio messages about other aircraft.
rdrsch	Performs search of the radar sector and determines if any detections are made.
rdrval	Calculates the incremental value to be added to the radar sector search value as a result of aircraft being in that radar sector.
remm	Removes dead missiles from the conscious pilot’s mental model.
rrsnsd	Forms observation for each aircraft on the observed list.
rwrobs	Constructs observations based upon the contents of the RWR device trackbank.

rwrsch	Performs search of the RWR sector and determines if any detections are made.
rwrval	Calculates the incremental value to be added to the RWR sector search value as a result of aircraft/missiles being in the RWR sector.
sectim	Initializes the sector search time history.
set_vis_sec	Determines the visual sectors in which targets are present.
tkupi	Initializes the state vector of the tracker for the first detection of an aircraft.
tkupmm	Updates the conscious pilot's mental model regarding the physical state of the detected aircraft.
tocobs	Performs external transfer of control observations.
upd_ifcmns	Updates the interface common blocks.
udsmvl	Increases the sector search values as a result of missiles being in that sector.
udsval	Increases the sector search values as a result of aircraft being in that sector.
visobs	Makes a list of visual observations and place them in memory.
vissch	Performs visual search of a visual sector and determines if any detections are made.
vismsl	Performs visual search and detect simulation for missiles.
visual	Performs visual search and detect simulation for aircraft.
visval	Calculates the incremental value to be added to a visual sector search value as a result of aircraft perceived to be in that visual sector.

The principal data structures (common blocks) involved in the maneuver selection process are described below.

/*extst*/	Stores the external status of all aircraft.
/*mind1*/	Stores mental model of sensor observations.
/*mind2*/	Holds value elements for each pilot.
/*mind3*/	Holds the pilot's assessment of relationships with other entities.
/*mind4*/	Holds mental model situational variables.
/*mindc*/	Stores mental model constants.
/*mindd*/	Stores information on aircraft believed to be dead.

/*mindlc*/	Holds mental model variables that are local to the current consciousness event.
/*mindms*/	Holds mental model perception of missiles.
/*mindpr*/	Holds production rule variables that influence pilot decisions.
/*mypfrm*/	Holds data about performance of decision maker's aircraft at the current time.
/*schang*/	Stores the significant changes list used to determine if a major situational update is required.
/*sencon*/	Holds constants used by sensor devices on aircraft.
/*sensed*/	Holds information on each detected aircraft during a single consciousness event.

Subroutine *conevt* is the executive that handles pilot consciousness events. It simulates pilot functions for one pilot at a time, including the situation update. After retrieving required common block data, *conevt* calls subroutine *perfrm* to update the pilot's knowledge of his aircraft's current performance parameters. The pilot's knowledge of his arcraft's physical state is updated later, through subroutine *mindup* which in turn calls *ccself*. *Conevt* uses subroutines *sensor* and *mindup* to update the pilot's knowledge and assessment of other platforms. *Sensor* makes the pilot's observations of other aircraft and *mindup* processes these observations to update and propagate aircraft and missile tracks in the pilot's mind.

Subroutine *sensor* is the executive routine for generating pilot observations of other aircraft and missiles. This is accomplished by explicitly simulating pilot observation of his different sensor displays as well as simulating direct visual observations through the cockpit canopy.

Once all observations have been made (through messages, sensors and inferred detections), subroutine *mindup* is called to merge the new observations with existing information in the conscious pilot's mind. *Mindup* first updates the pilot's knowledge of his ownship by calling subroutine *ccself*. Subroutine *cc2x0* is then called to process new observations on other aircraft. After performing initialization tasks, *cc2x0* loops through each observation. A check is made to determine if the observation is worth processing by calling subroutine *cc2x1*. If the observation is a first detection, *cc2x3* is called to add the observation to the conscious pilot's mental model. If the observation was not a first detection, subroutine *cc2x2* is called to update the pilot's perception with the new information. If significant changes have occurred to an aircraft, such as a newly detected aircraft, messages are prepared to be sent to flight mates relaying the significant information.

Upon completion of the situation update, subroutine *conevt* calls subroutine *modsel* to select a pilot action based on the updated mental model. Radio messages to flight mates are then sent out and a call to *makece* schedules the next consciousness event for this pilot.

Subroutine CONEVT

Subroutine *conevt* is the executive routine for processing consciousness events. The pilot model situation update is one of the functions that *conevt* performs. Figure 2.15-2 is the functional flow diagram that describes the logic used to implement *conevt*. The blocks are numbered for ease of reference in the following discussion.

Block 1 Subroutine *inlsvd* is called to retrieve data specific to this consciousness event.

Block 2 Test if this aircraft is under Brawler control, or whether it is being controlled by a manned simulator or some other external process. If the aircraft is not under Brawler control, return.

Block 3 Test whether the conscious pilot is dead or alive. If dead, jump to Block 26.

Block 4 Test if this consciousness event is obsolete. If so, jump to Block 26. A consciousness event will become obsolete if, after it has been scheduled, some other event causes the scheduling of another consciousness event for the same pilot at an earlier time than the one already scheduled.

Block 5 Subroutine *mindin* is called to retrieve the data for the pilot's mental model.

Block 6 Subroutines *grdrs, grdrc and gfcsta* are called to retrieve radar status, radar characteristics, and fire control status for the current aircraft.

Block 7 Subroutine *flyac* is called to update the state vector for this aircraft.

Block 8 Check that the aircraft has not flown into the ground. If it has, jump to Block 26.

Block 9 Test if the user has disabled consciousness events for this aircraft. This is actually a two-step test. If the *cev_off* flag is set, indicating that consciousness events are disabled, a call is made to the production rules to give the user a final chance to unset the flag. If it remains set, then jump to Block 24.

Block 10 Subroutine *perfrm* is called to update the pilot's perception of his aircraft's current performance capabilities.

Block 11 Subroutine *fncomi* is called to initialize messages to be sent out by the conscious pilot.

Block 12 Subroutine *sensor* is called to make visual and sensor observations. This routine is detailed below.

Block 13 Test if the consciousness event type is equal to the inferred detection type (7). If true, continue at Block 14. If false, skip to Block 15.

Block 14 Entry point *inferl* of subroutine *inferd* is called to assess inferred aircraft detections. *Inferl* is called when a previously unobserved aircraft has just killed a friendly aircraft.

Block 15 Subroutine *mindup* is called to update the mental model of the currently conscious pilot based upon receipt of radio messages and new observation information. This routine is detailed below.

Block 16 Subroutine *modsel* is called to select an action based on the updated mental model.

Block 17 Subroutine *fncom* is called to send messages prepared for flight mates.

Block 18 Test whether the conscious pilot has a GCIAWACS controller. If true, continue at Block 19. If false, go to Block 20.

Block 19 Subroutine *chk_ammo_lev* is called to check to see if an out of ammo message is needed.

Block 20 Test if this is an initial consciousness event for a DLI (deck launched interceptor). If true, continue at Block 21. If false, skip to Block 22.

Block 21 Subroutine *pmsdga* is called to send a message to an AWACS.

Block 22 Check if any messages have been received. If yes, continue at Block 23. If no, skip to Block 24.

Block 23 Subroutine *dlmstr* is called to delete received messages. Note that the messages would have already been read in subroutine *mindup*, so they are not discarded before being read.

Block 24 Subroutine *gcetim* is called to determine the time interval to the next self-planted consciousness event.

Block 25 Subroutine *makece* is called to plant the next consciousness event in the event list.

Block 26 Subroutine *upd_ifcmns* is called to update the interface common blocks. This routine is only functional in manned simulator applications and is a null stub for all other installations of Brawler.

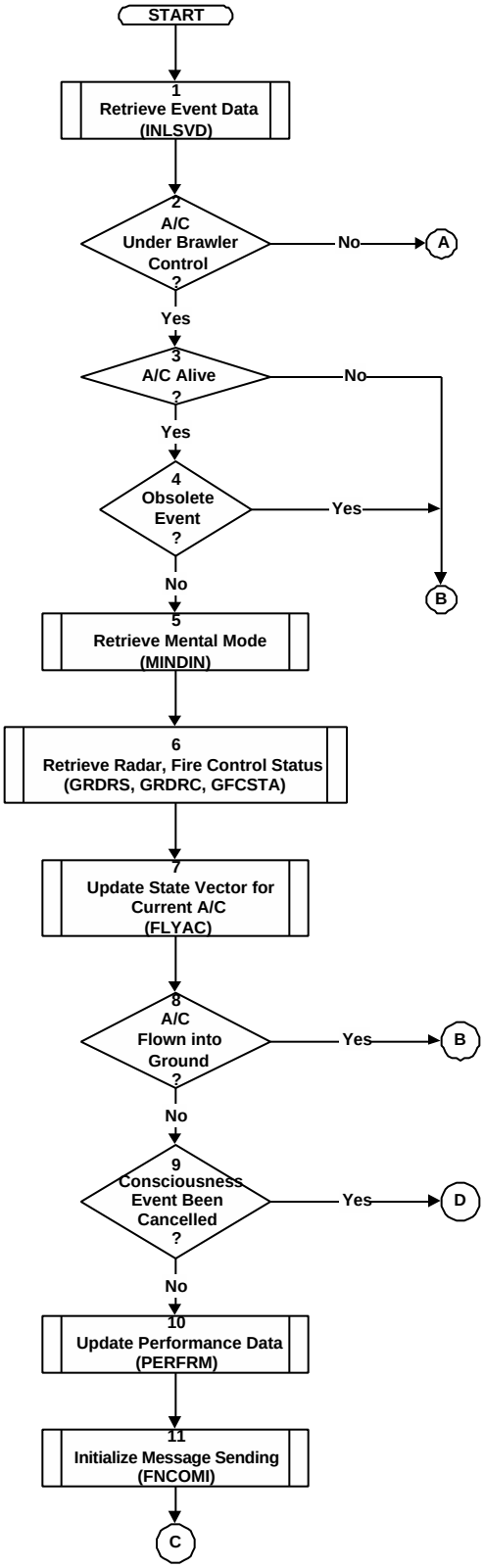

FIGURE 2.15-2. CONEVT Functional Flow Diagram (Page 1 of 3).

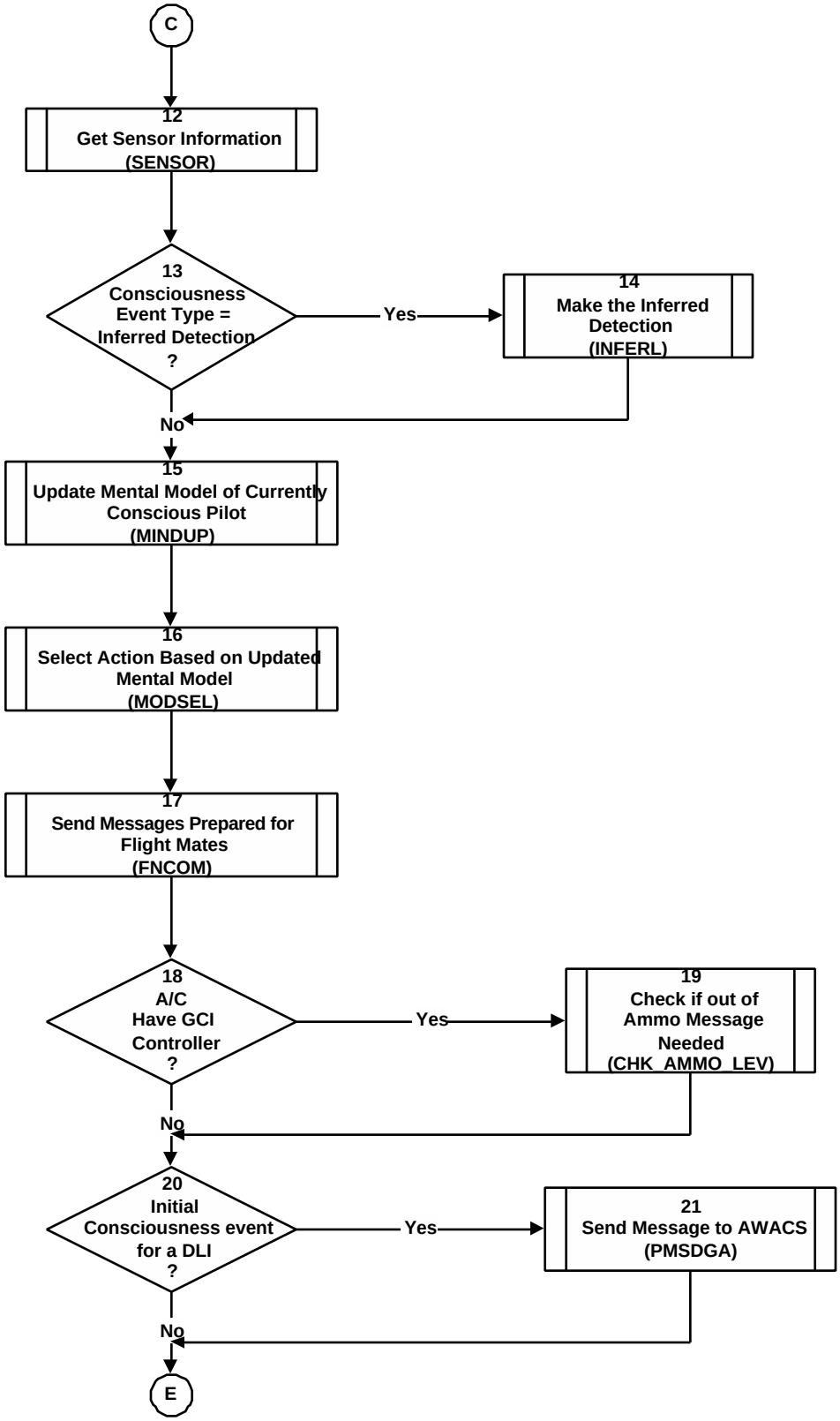

FIGURE 2.15-2. CONEVT Functional Flow Diagram (Page 2 of 3).

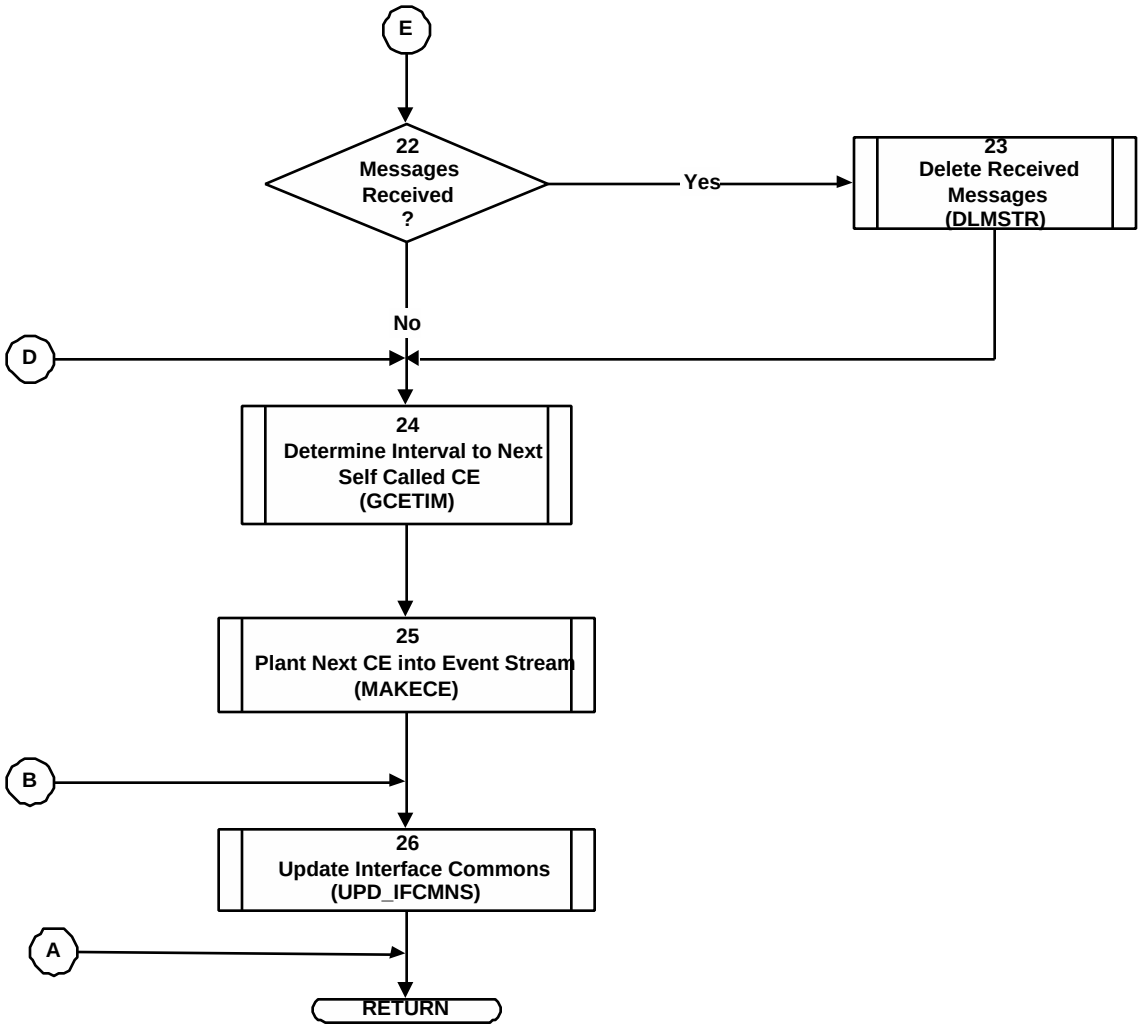

FIGURE 2.15-2. CONEVT Functional Flow Diagram (Page 3 of 3).

Subroutine PERFRM

Subroutine *perfrm* computes aircraft performance variables. Figure 2.15-3 is the functional flow diagram that describes the logic used to implement *perfrm*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Initialize variables for the current aircraft. This includes transferring previously calculated values from aircraft status variables to pilot mental model variables as well as calculating new values for corner velocity and min and max instantaneous Gs base upon current aircraft state.

Block 2. Test if the entity's mach number is precisely zero. If the mach number is zero, assume that the entity is a SAM site and set the maximum sustained Gs, drag versus lift tables and angle of attack versus lift tables to zero. For a non zero mach number, calculate

these values (see Blocks 4-6 below). *Blocks 2 and 3 are a modification that will be available in version 6.3. Version 6.2 executes Blocks 4-6 unconditionally.*

Block 3. For the case where the entity's mach number is zero (entity assumed to be a SAM site), set the maximum sustained Gs, drag versus lift tables and angle of attack versus lift tables equal to zero.

Block 4. For the case where the mach number is non zero (an aircraft), calculate a value for maximum sustained Gs (*gmxsut*) as a function of mach, current altitude, dynamic pressure, aircraft weight and throttle setting.

Blocks 5-6 are in a loop over the number of entries in the drag versus lift and angle of attack versus lift tables.

Block 5. Calculate drag versus lift and angle of attack versus lift table entry as a function of mach, dynamic pressure and lift. These are computed via calls to functions *drag* and *aoa*.

Block 6. Test if there are more entries in the tables remaining to be processed. If so, go back to Block 5. If not, continue with Block 7.

Blocks 7-13 calculate the maximum G value (*gmxsu*) which is used later to determine the maximum turn rate. The *gmxsu* is not intended as the true maximum sustained G capability. It is computed for use in later planning and represents the maximum Gs that the pilot will use in any situation except extreme emergency.

Block 7. Test if the aircraft has a firing interest. If the missile mode (*mslmd*) is equal to 1, 2 or 3 then the aircraft has a firing interest and Block 8 is executed.

Block 8. Set the maximum Gs (*gmxsu*) equal to the maximum load possible at maximum thrust.

Block 9. Test if there are any hostile aircraft or hostile missiles nearby (<20 nmi). If no hostiles are in the vicinity, then there is no need for hard turns and Block 10 is executed.

Block 10. Set the maximum Gs (*gmxsu*) equal to the maximum sustained G (*gmxsut*) value calculated previously (Block 4).

Block 11. Test if the aircraft is travelling faster than Mach 1.0 or more than 100 feet/sec faster than its corner velocity. If either condition is true, go to Block 12. If not, go to Block 13. *There is an error in the Mach test in this code in version V6.2 that has been corrected in version V6.3.*

Block 12. Hostile are in the vicinity and you are going fast enough to turn at the aircraft's maximum load, so set maximum Gs (*gmxsu*) to the maximum load at maximum thrust (*gmxin*).

Block 13. Set maximum Gs (*gmxsu*) to the maximum sustained G value (*gmxsut*).

Blocks 14 and 15 are a modification that will be available in version 6.3. Version 6.2 executes Block 16 unconditionally.

Block 14. Test for zero speed. If so, assume that this is a SAM site and go to Block 15. If not, go to Block 16.

Block 15. The maximum turn rate (*wmax*) and drag at maximum sustained Gs (*drgsu*) are zeroed for SAM sites.

Block 16. The maximum turn rate (*wmax*) is calculated from maximum Gs (*gmxsu*) and aircraft speed. The drag at maximum sustained Gs is calculated using values from the drag versus lift tables computed in Block 5.

Block 17. Subroutine *effthr* is called to calculate the effective gross thrust and effective propulsive drag for the current state of the aircraft.

Block 18. Subroutine *effthr* is called to calculate the maximum effective gross thrust and the maximum effective propulsive drag.

Block 19. Subroutine *effthr* is called to calculate the minimum effective gross thrust and the minimum effective propulsive drag.

Block 20. The current drag, minimum and maximum longitudinal acceleration are calculated.

Block 21. The minimum and maximum allowable angles of attack are calculated.

Blocks 22-23 are in a loop over the number of entries in the drag versus angle of attack tables.

Block 22. Calculates a drag versus angle of attack table entry.

Block 23. If table not yet filled, go back to Block 22. If table finished, continue to Block 24.

Blocks 24-28 are in a loop over the number of throttle settings in the longitudinal force versus throttle tables. There are 5 throttle settings:

1. drag devices deployed
2. damaged aircraft, set to small negative value
3. idle
4. military thrust
5. full afterburner

Block 24. Subroutine *thrust* is called to calculate the gross thrust due to gas generator and afterburner output.

Block 25. Test if drag devices are deployed. If deployed, go to Block 26. If not, go to Block 27.

Block 26. Drag devices have been deployed, calculate the drag due to drag devices via a call to subroutine *fdddrg*.

Block 27. Compute the longitudinal force for this throttle setting as the sum of the thrust from the gas generator and afterburner minus any drag from the drag devices.

Block 28. Test to compute another table entry if the table is not full yet. If table not finished, go to Block 24. If finished, return.

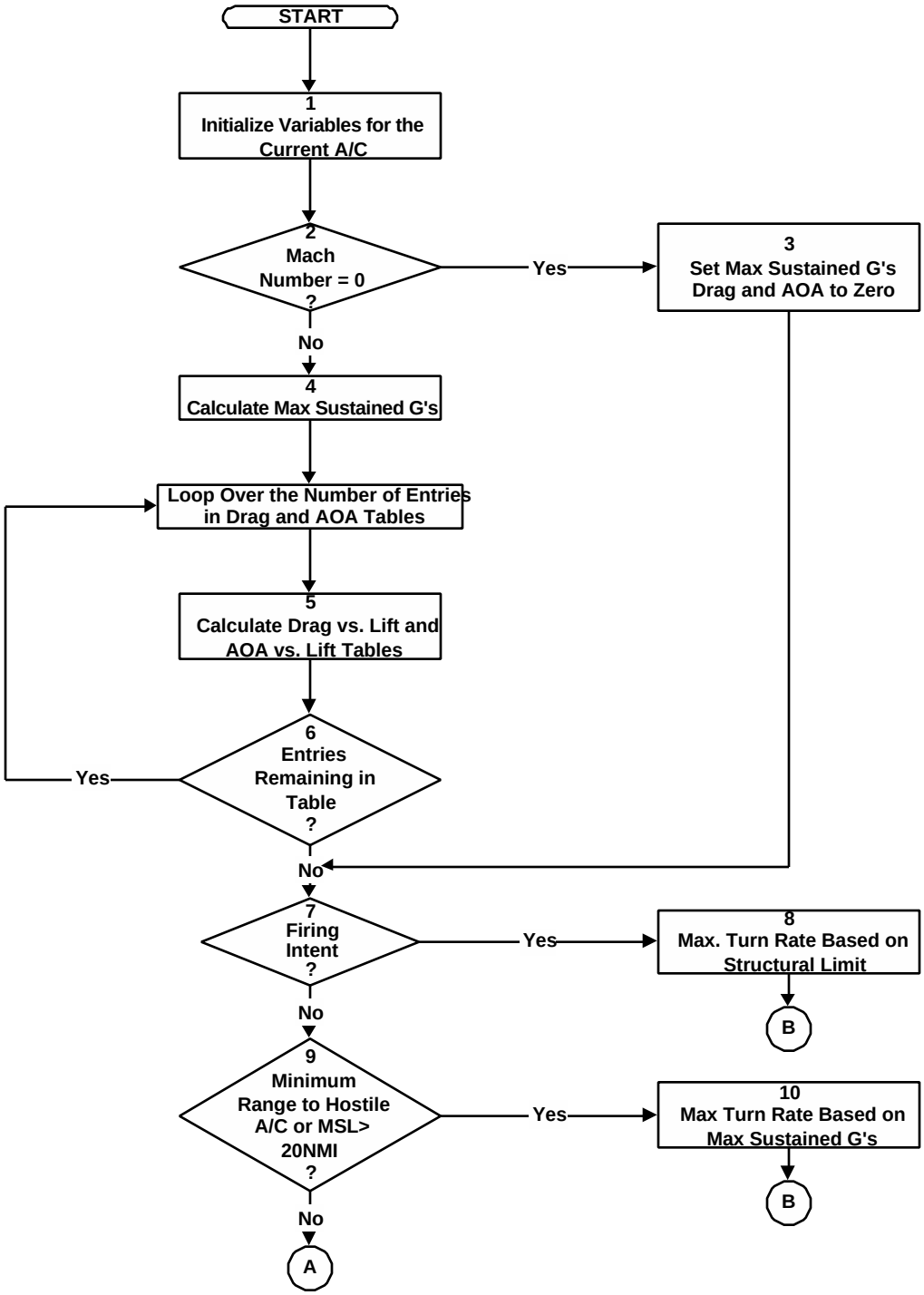

FIGURE 2.15-3. PERFRM Functional Flow Diagram (Page 1 of 3).

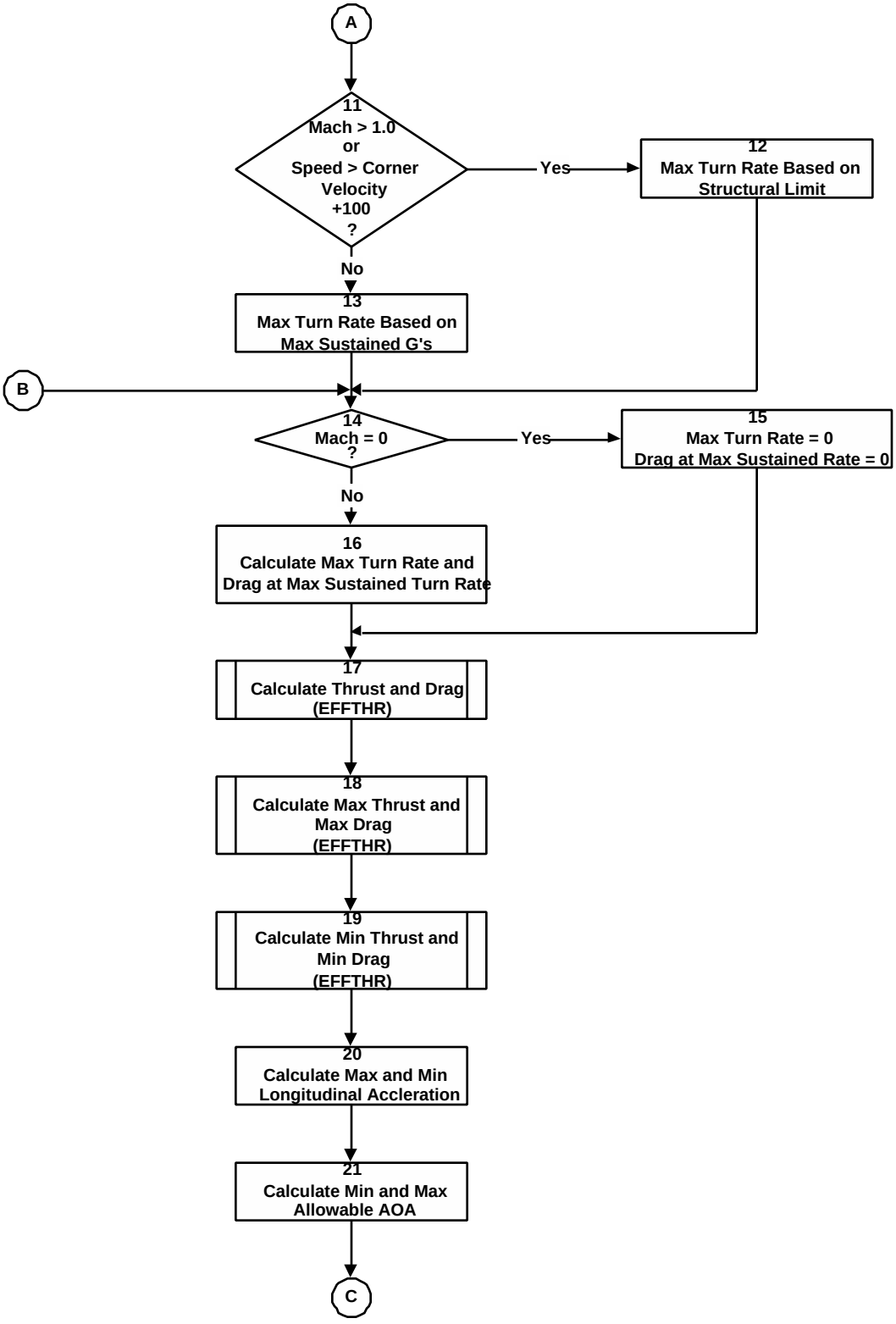

FIGURE 2.15-3. PERFRM Functional Flow Diagram (Page 2 of 3).

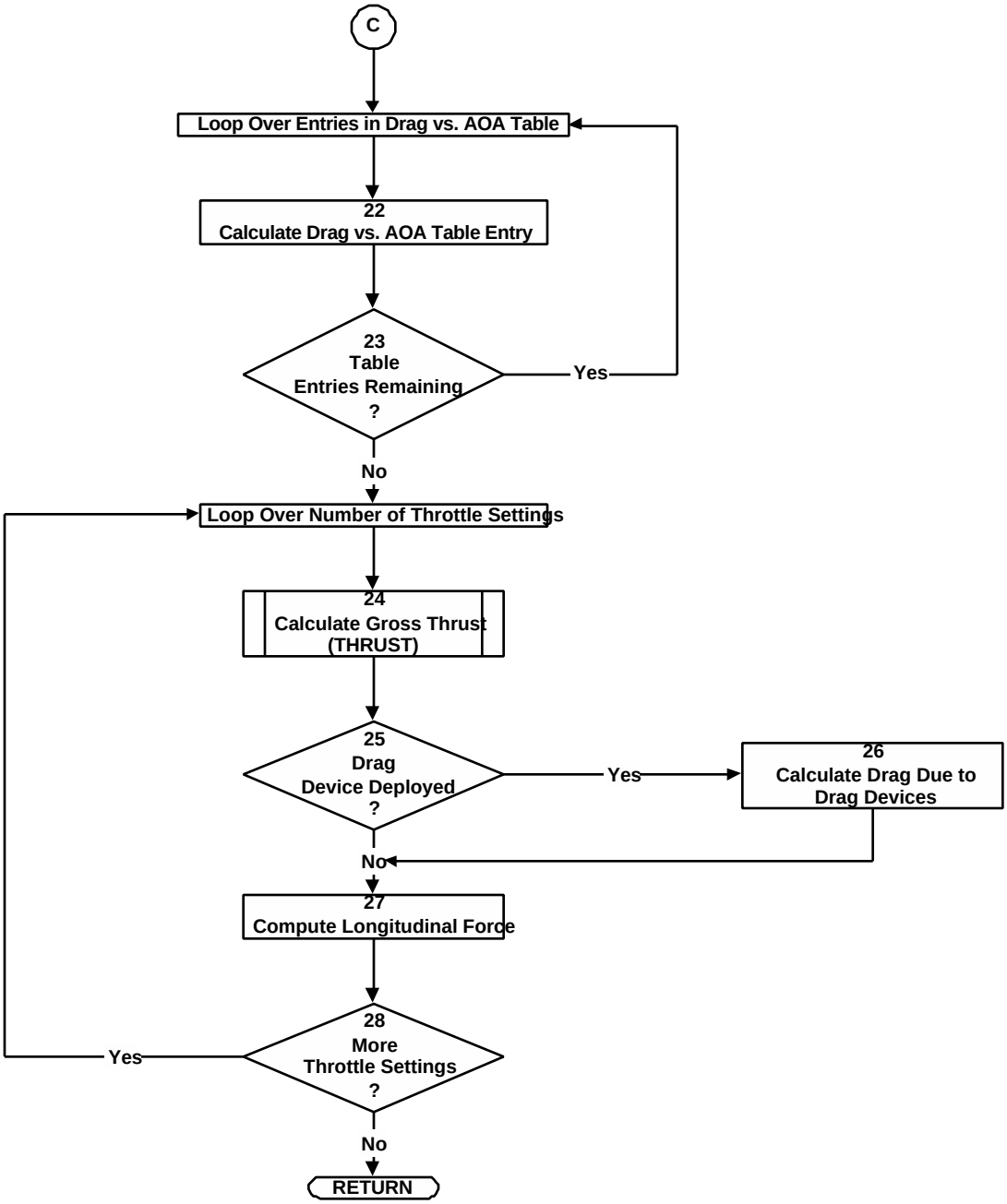

FIGURE 2.15-3. PERFRM Functional Flow Diagram (Page 3 of 3).

Subroutine SENSOR

Subroutine *sensor* is an executive routine for conducting sensor searches. Figure 2.15-4 is the functional flow diagram that describes the logic used to implement *sensor*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Initialize common blocks and local arrays.

Block 2. Subroutine *a_lim* is called to limit the acceleration vector to avoid negative speeds during the projection of aircraft trajectories.

Block 3. Subroutine *projw* is called to project the aircraft to the current simulation time.

Block 4. Test if this is the first consciousness event for this aircraft. If true, go to Block 5. If false, go to Block 6.

Block 5. Subroutine *sectim* is called to initialize the sector search time history.

Block 6. Subroutines *setsvl* is called to initialize the sector search priority values for aircraft and missile search (this routine is detailed below).

Block 7. Subroutine *acnear* is called to adjust the sector search priority values if aircraft are present in that sector (this routine is detailed below).

Block 8. Subroutine *mslner* is called to adjust the sector search priority values if missiles are present in that sector.

Block 9. Test to see if this consciousness event is associated with the transfer of control of this aircraft into Brawler from another simulation, which can happen when running in confederation with other simulations. If so, go to Block 10. If not, go to Block 11.

Block 10. Subroutine *tocobs* is called to create external transfer of control "observations." These are used to initialize the pilot's awareness of other players at the instant of the transfer of control.

Block 11. Subroutine *set_vis_sec* is called to determine which visual sectors may contain targets.

Blocks 12 through 14 are in a loop over the number of search sectors.

Block 12. Subroutine *sctsch* is called to search individual sectors from the highest valued sectors to the lowest valued sectors (this routine is detailed below).

Block 13 Test if any search time remains. If true, continue to Block 14. If false, jump to Block 15.

Block 14. Test if any sectors remain to be searched. If true, go back to Block 12. Otherwise, end the searching of sectors and proceed to block 15.

Block 15. Test if aircraft has a functioning RHAW device. If true, go to Block 16. If false, go to Block 17. *This is an obsolete feature. This flag should never be true.*

Block 16. Subroutine *rdrhaw* is called to update the pilot's perception of RHAW detections.

Block 17. Subroutine *pcode* is called to perform any user defined manipulations of the observations in the productions rules. The intent here is to allow the user to enhance or degrade the pilot's ability to learn or infer the type of an observed aircraft rather than creating or deleting the observations themselves.

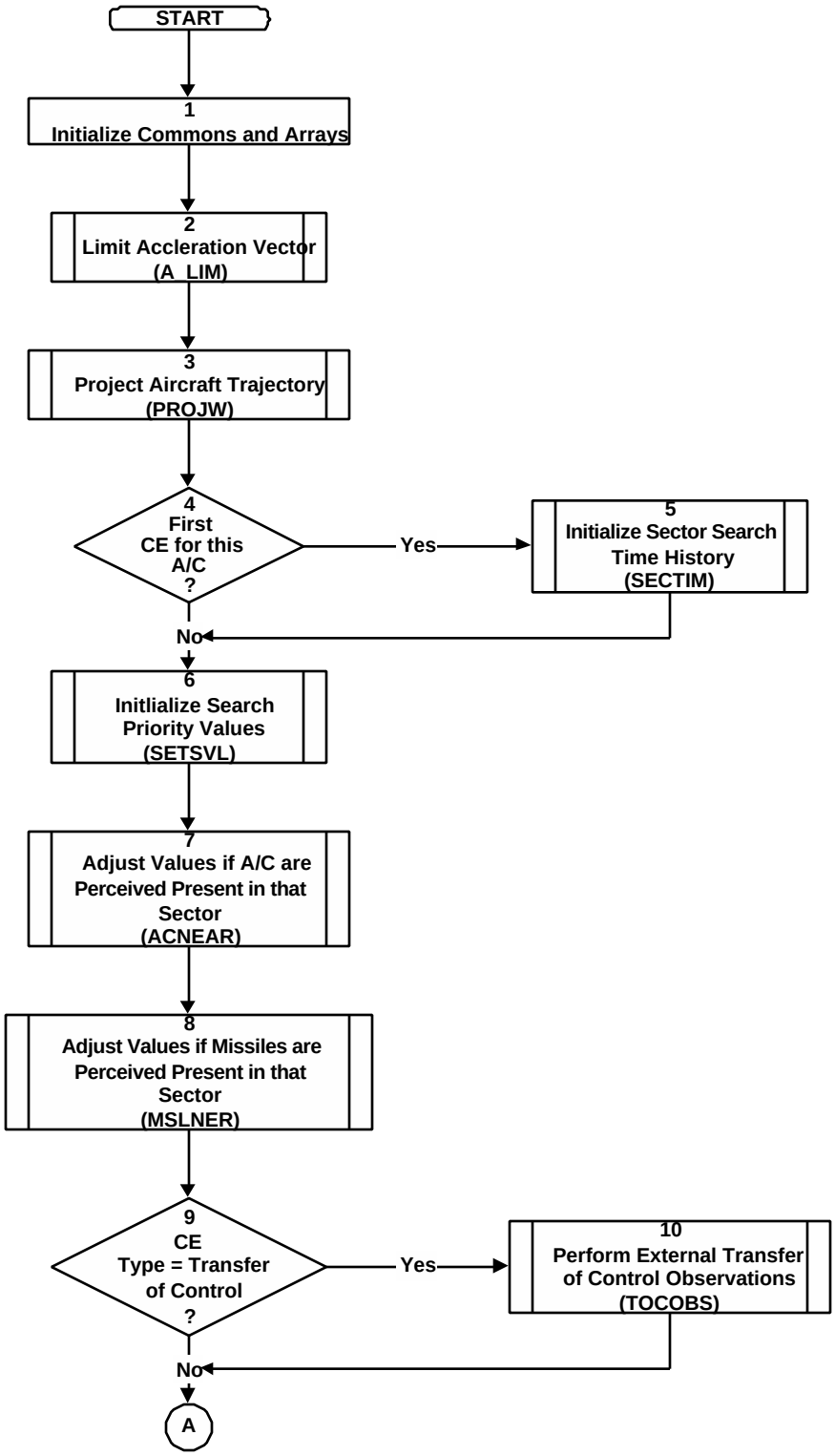

FIGURE 2.15-4. SENSOR Functional Flow Diagram (Page 1 of 2).

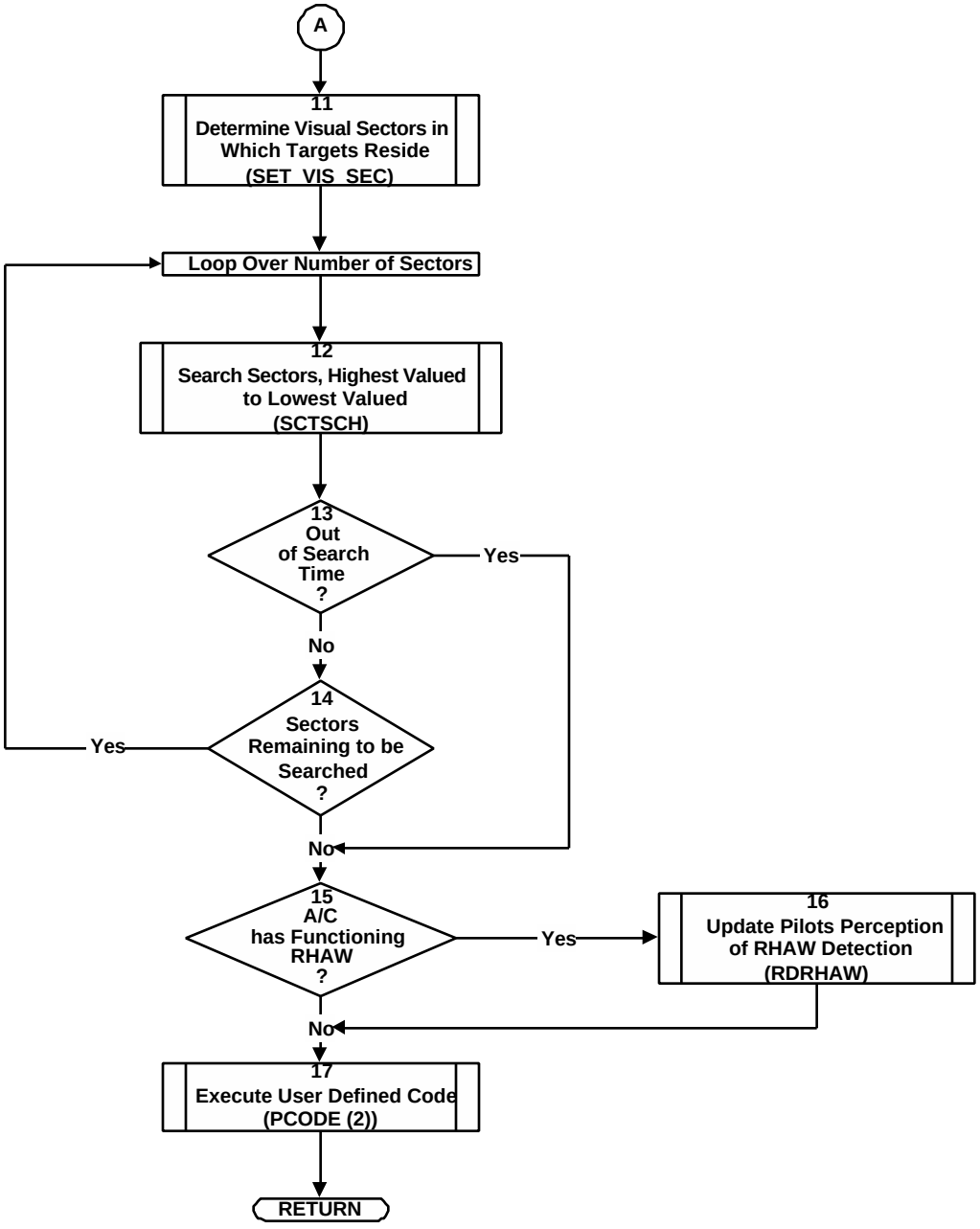

FIGURE 2.15-4. SENSOR Functional Flow Diagram (Page 2 of 2).

Subroutine SETSVL

Subroutine *setsvl* initializes sector search values before each search on the basis of the time elapsed since the search of each sector and user input parameters for each sector, that indicate their relative importance. Figure 2.15-5 is the functional flow diagram that describes the logic used to implement *setsvl*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Initialize sector search values using values read in from the **SENCON** file during initialization.

Block 2. Test if this is the first consciousness event for this pilot. If true, jump to Block 4, so as to not inhibit visual search sector number 5. If false, go to Block 3. This sector is below and to the rear of the pilot and is normally blocked by the body of the aircraft. Search of this sector is allowed on the first pass only, to allow the pilot to view any flight mates in that area. At all other times visual search sector 5 is inhibited.

Block 3. Inhibit visual search sector 5 by setting its value to zero.

Block 4. Test if the aircraft has a sensor fusion device. If true, continue at Block 5. If false, jump to Block 6.

Block 5. Zero sector search values for all avionics except the sensor fusion device (sectors 9-19, excluding sector 15 are zeroed). Jump to Block 23.

Blocks 6-22 are executed if the aircraft does not have a sensor fusion device. Checks are made for the existence of other avionics devices and sector search values are zeroed for the appropriate sector if that device does not exist or is not functional.

Block 6. Set the sector search value for the integrated display (SFD) equal to zero, since it does not exist on this aircraft.

Block 7. Test whether the aircraft has any radar antennas turned on. If no, go to Block 8. If yes, jump to Block 9.

Block 8. The sector search value for the radar sector is set to zero, since no antennas are presently turned on.

Block 9. Test if the aircraft has an infrared search and track device (IRST). If no, go to Block 10. If yes, jump to Block 11.

Block 10. The sector search value for the IRST sector is set to zero, since no IRST device is present.

Block 11. Test if the aircraft has a radar warning receiver (RWR). If no, go to Block 12. If yes, jump to Block 13.

Block 12. The sector search value for the RWR sector is set to zero, since no RWR device is present.

Block 13. Test if the aircraft has a missile approach warning (MAW) device. If no, go to Block 14. If yes, jump to Block 15.

Block 14. The sector search value for the MAW sector is set to zero, since no MAW device is present.

Block 15. Test if the aircraft has a missile warning (MW) device. If no, go to Block 16. If yes, jump to Block 17.

Block 16. The sector search value for the MW sector is set to zero, since no MW device is present.

Blocks 17-20 are new additions for version V6.3. For version V6.2, jump to Block 21.

Block 17. Test if the aircraft has a electronic support measure (ESM) device. If no, go to Block 18. If yes, jump to Block 19.

Block 18. The sector search value for the ESM sector is set to zero, since no ESM device is present. Jump to Block 21.

Block 19. Test if the aircraft has any active ESM fields of view (FOV). If no, go to Block 20. If yes, jump to Block 21.

Block 20. The sector search value for the ESM sector is set to zero, since no ESM fields of view are active.

Block 21. The sector search values for the reserved sectors (16-18) are set to zero.

Block 22. The initial sector search value for the identify friend or foe (IFF) sector is set to zero, since you only want to look at this device to identify a target. Modifiers to this sector search value may be applied through subroutines that adjust values based on other aircraft being present.

Blocks 23-27 loop over the number of search sectors to modify sector values based on the time since the last search of that sector.

Block 23. Test the desired time between searches for this sector. If not zero, go to Block 24. If zero, go to Block 25. Desired revisit times for each sector are read in from the **SENCON** file during initialization.

Block 24. Sector search values are modified using a one minus a Cauchy function of the time since the last observation of this sector, scaled by the desired revisit time.

Block 25. Test if the current sector is a visual sector (1-8). If so, go to Block 26. If not, jump to Block 27.

Block 26. Add random jitter to the sector value to break ties among the 8 visual sectors. Jitter values range from 0.0 to 0.001.

Block 27. Test if there are any more sectors remaining to be processed in this loop. If true, process the next sector (jump back to block 23). If false, return.

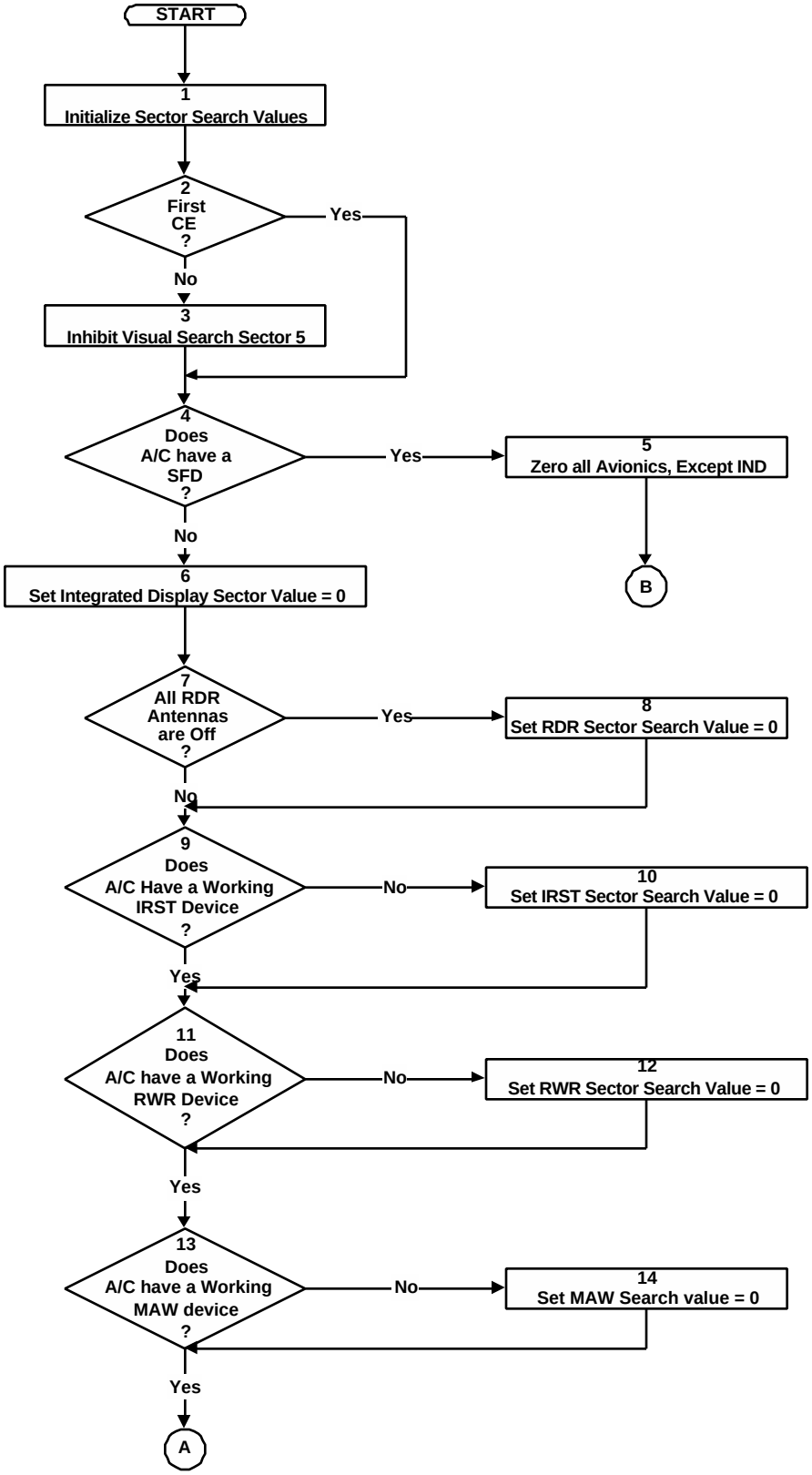

FIGURE 2.15-5. SETSVL Functional Flow Diagram (Page 1 of 2).

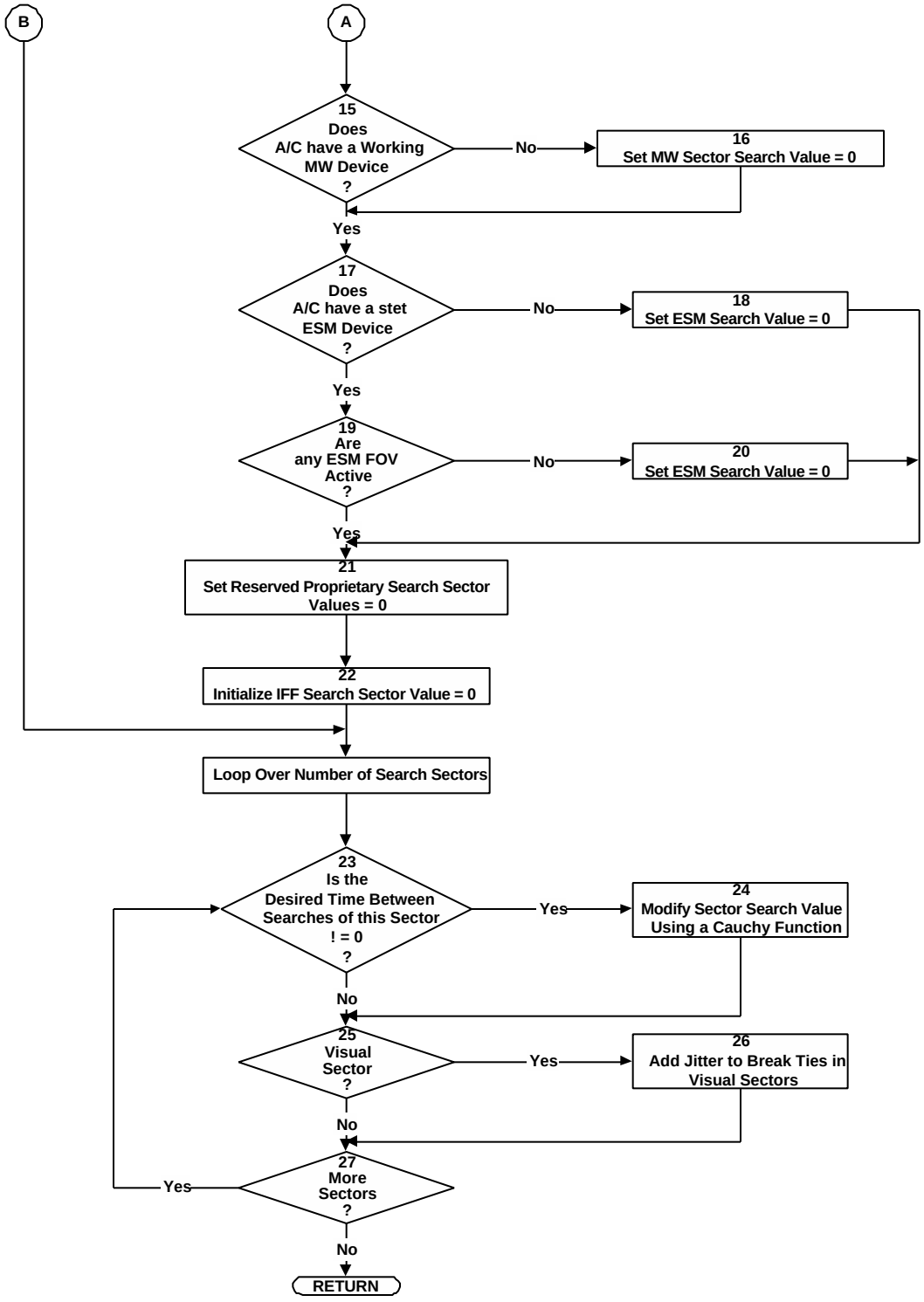

FIGURE 2.15-5. SETSVL Functional Flow Diagram (Page 2 of 2).

Subroutine ACNEAR

Subroutine *acnear* counts and identifies aircraft that the pilot believes he may see in each observation sector and then updates the sector search values accordingly. Figure 2.15-6 is the functional flow diagram that describes the logic used to implement *acnear*. The blocks are numbered for ease of reference in the following discussion.

Blocks 1-29 are in a loop over the number of aircraft in the detailed consideration group for the currently conscious pilot.

Block 1. Test if the aircraft being checked is the conscious pilot's aircraft. If true, increment loop to next aircraft on list.

Block 2. Subroutine *prjacc* is called to make a constant acceleration projection of the aircraft checked to bring it up to the current simulation time.

Block 3. The line of sight (LOS) vector and line of sight range are calculated via calls to vector utilities *vsub* and *xmag*.

Block 4. Subroutine *getsec* is called to calculate which search sectors the aircraft under consideration resides in. It sets a flag for each sensor sector that the aircraft resides in. If the range to the target aircraft is less than 50,000 feet, getsec will also return the number of the visual sector (1-8) containing the target.

Block 5. Test if the aircraft under consideration is in one of the eight visual sectors. If true, go to Block 6. If not, go to Block 8.

Block 6. Subroutine *incsec* is called to update the list of aircraft in the visual search sector.

Block 7. Subroutine *udsval* is called to increase the visual sector search value as a result of an aircraft being in that sector.

Block 8. Test if the aircraft under consideration is in the radar sector. If true, go to Block 9. If false, jump to Block 12.

Block 9. Subroutines *grdrs* and *grdrc* are called to make the radar status and radar characteristics data current.

Block 10. Subroutine *incsec* is called to update the list of aircraft in the radar search sector.

Block 11. Subroutine *udsval* is called to increase the radar sector search value as a result of an aircraft being in that sector.

Block 12. Test if the aircraft under consideration is in the infrared search and track (IRST) sector. If true, go to Block 13. If false, go to Block 15

Block 13. Subroutine *incsec* is called to update the list of aircraft in the IRST search sector.

Block 14. Subroutine *udsval* is called to increase the IRST sector search value as a result of an aircraft being in that sector.

Block 15. Test if the aircraft under consideration is in the identify friend or foe (IFF) sector. If true, go to Block 16. If false, jump to Block 23.

Block 16. Subroutine *incsec* is called to update the list of aircraft in the IFF search sector.

Blocks 17-22 are contained within a loop over all IFF devices on the conscious pilot's aircraft.

Block 17. Subroutine *giffs* is called to load the next IFF device into memory.

Block 18. Subroutine *giffc* is called to update the IFF characteristics data.

Block 19. Test if IFF data was requested in the previous consciousness event. If true, go to Block 20. If false, go to Block 21.

Block 20. Set the IFF array value to indicate that an aircraft may be observed by this IFF device. The highest scoring device will later be the one search by the pilot during the search phase, time permitting.

Block 21. Subroutine *udsval* is called to increase the IFF sector search value as a result of an aircraft being in that sector. Following this, the value of the variable *prffmd* is tested to allow the user to turn this IFF device off via production rules.

Block 22. Test if there are more IFF devices to be processed, if true, process the next IFF device (go back to Block 17).

Block 23. Test if the aircraft under consideration is in the radar warning receiver (RWR) sector. If true, go to Block 24. If false, go to Block 26.

Block 24. Subroutine *incsec* is called to update the list of aircraft in the RWR search sector.

Block 25. Subroutine *udsval* is called to increase the RWR sector search value as a result of an aircraft being in that sector.

Block 26. Test if the aircraft under consideration is in the integrated display (IND) sector. If true, go to Block 27. If false, go to Block 29.

Block 27. Subroutine *incsec* is called to update the list of aircraft in the IND search sector.

Block 28. Subroutine *udsval* is called to increase the IND sector search value as a result of an aircraft being in that sector.

Block 29. Test if there are any more aircraft in the detailed consideration list. If true, process the next aircraft in the list (jump back to Block 1).

Block 30. Determine the most valuable IFF device as scored from the number of aircraft each IFF device detected (Blocks 16-22).

Block 31. Put SAMs that are in the same flight on each others' nearby list for the first consciousness event only. This guarantees that the SAMs will know about each other, which should be the case if they are in the same flight.

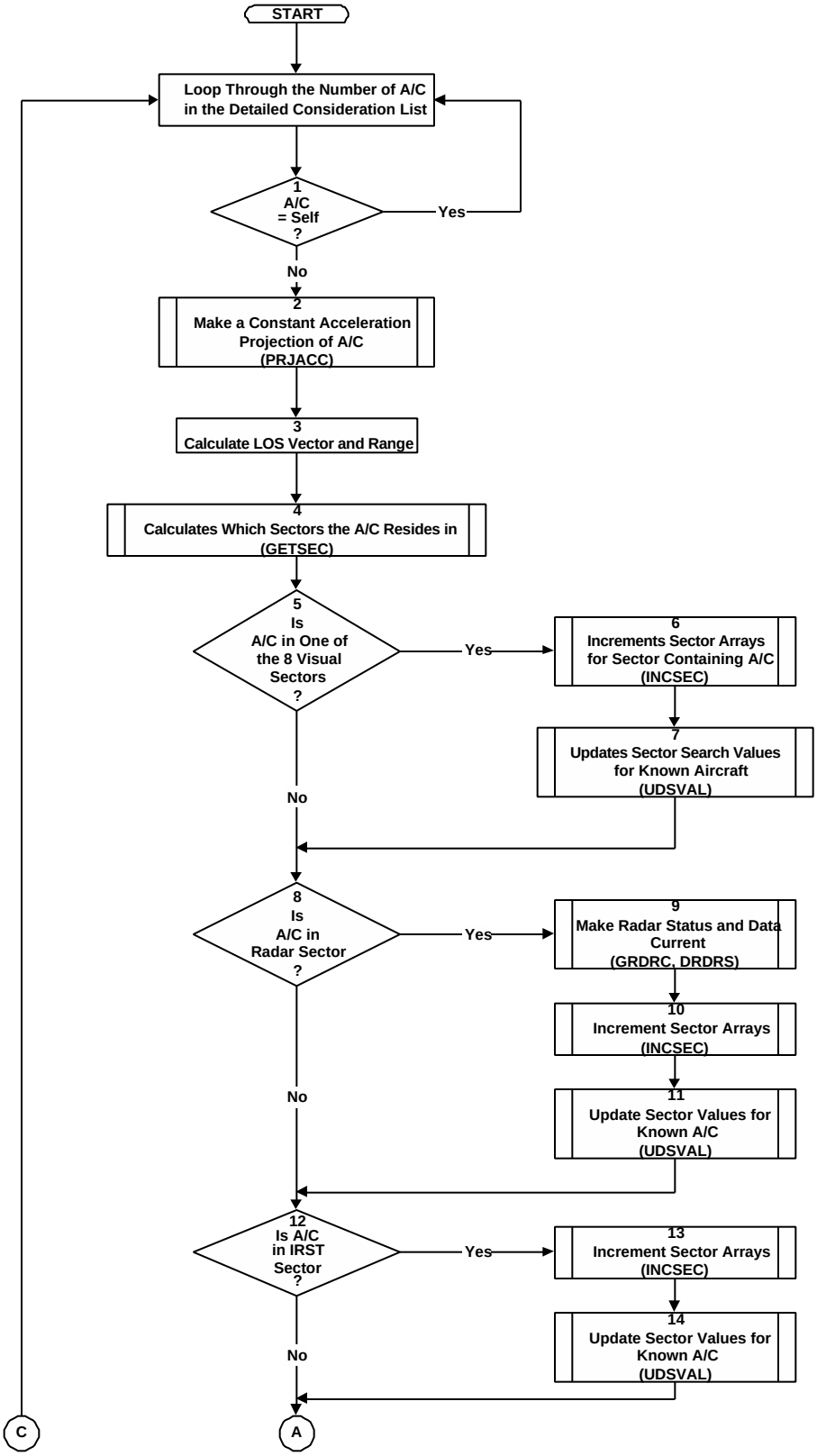

FIGURE 2.15-6. ACNEAR Functional Flow Diagram (Page 1 of 3).

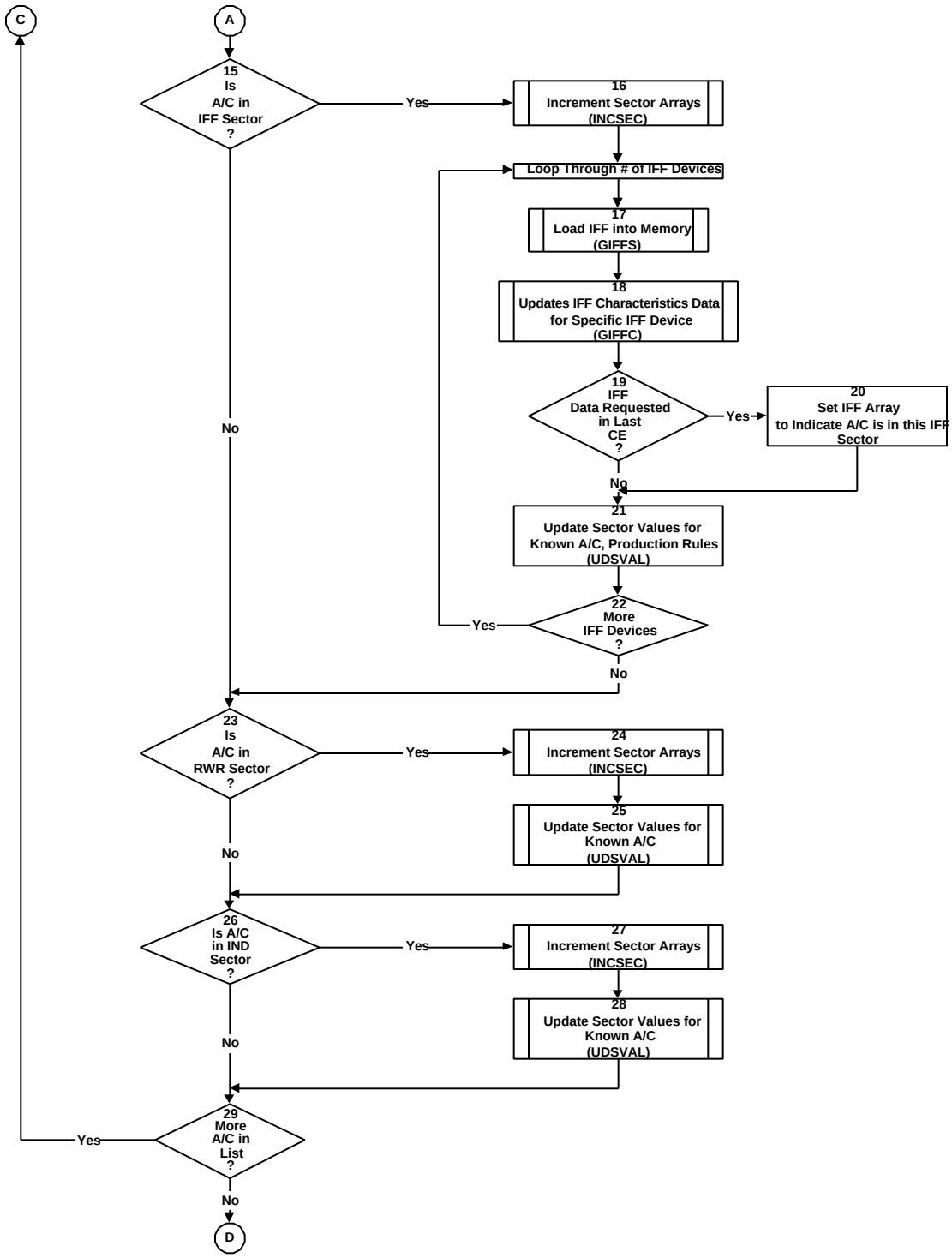

FIGURE 2.15-6. ACNEAR Functional Flow Diagram (Page 2 of 3).

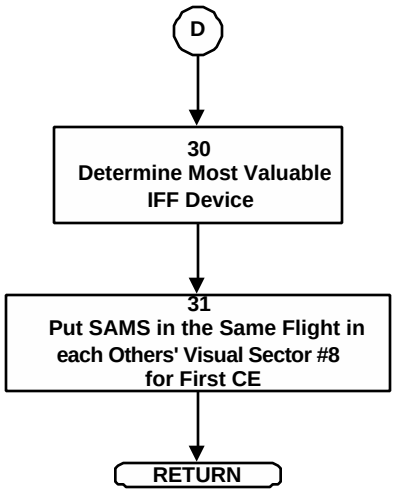

FIGURE 2.15-6. ACNEAR Functional Flow Diagram (Page 3 of 3).

Subroutine RDRVAL

Subroutine *rdrval* calculates the incremental value to be added to the radar sector search value as a result of aircraft being in the radar sector. Figure 2.15-7 is the functional flow diagram that describes the logic used to implement *rdrval*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Test if the target aircraft is in the conscious pilot's mental model. If not, then return to the calling routine (udsval), otherwise continue to Block 2.

Block 2. Test if the target aircraft is known to be a friendly aircraft. If yes, go to Block 3. If no, go to Block 6.

Block 3. Set the time constant for observation to every 10-15 seconds, based on proximity to conscious pilot. Friendly aircraft that are close to the conscious pilot are set for observation every 10 seconds. The time constant increases with distance from the conscious pilot, up for observation up to 15 seconds for friendly aircraft at extreme range.

Block 4. An importance value of 0.5 is assigned to all friendly aircraft.

Block 5. The incremental sector search value is calculated as the importance value times one minus a Cauchy function, with time since last observation and the time constant (Block 3) as the driving parameters of the Cauch function. After this, return.

Block 6. This block is the beginning for the calculation of the sector search value for hostile and unknown aircraft. The importance variable is calculated as a function of engagement utility, value of self and the last computed value for engaging the hostile aircraft.

Block 7. Subroutine *asstgt* is called to retrieve the conscious pilot's target assignment from his flight leader.

Block 8. Test if the aircraft under consideration is the target assigned by the flight leader. If true, go to Block 9. If false, jump to Block 10.

Block 9. The importance variable, calculated in Block 6, is incremented by 0.5 due to the aircraft being the assigned target of the conscious pilot.

Block 10. Test if the aircraft is the selected target of the conscious pilot. If true, go to Block 11. If false, jump to Block 12.

Block 11. The importance variable, calculated in Block 6, is incremented by 0.5 due to the target aircraft being the selected target of the conscious pilot.

Block 12. Set the time constant for observation of the target aircraft at every 2-10 seconds, based on its proximity to the conscious pilot. Hostile or unknown aircraft very close to the conscious pilot's aircraft are set for an observation every 2 seconds. The time constant increases with distance from the conscious pilot, up to 10 seconds for hostile aircraft at extreme range.

Block 13. The incremental sector search value is calculated as the importance value times one minus a Cauchy function, with the time since the last observation and the time constant (Block 12) as the driving parameters of the Cauchy function.

Block 14. Test if the conscious pilot is trying to achieve single target track (STT) on the hostile target aircraft and that the target is in the field of the antenna that he is trying to lock. If true, go to Block 15. If false, return.

Block 15. Award additional value to the incremental sector search value if the target is likely to remain in the antenna FOR for the next 10 seconds and the radar sector has not been looked at for at least one second. This is computed using a border function based on range plus range rate times 10 seconds, multiplied by a Cauchy function based on time since last radar search.

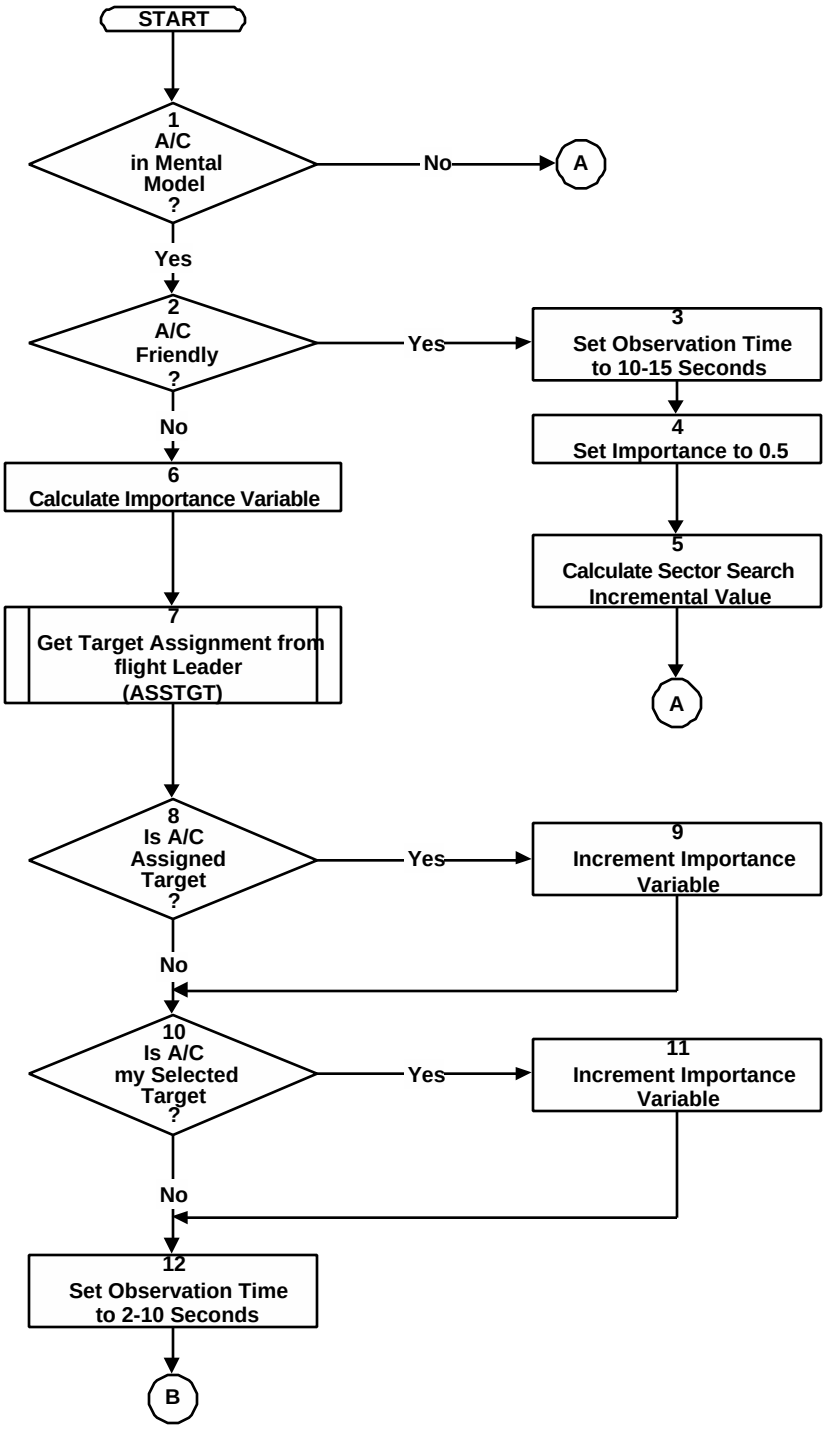

FIGURE 2.15-7. RDRVAL Functional Flow Diagram (Page 1 of 2).

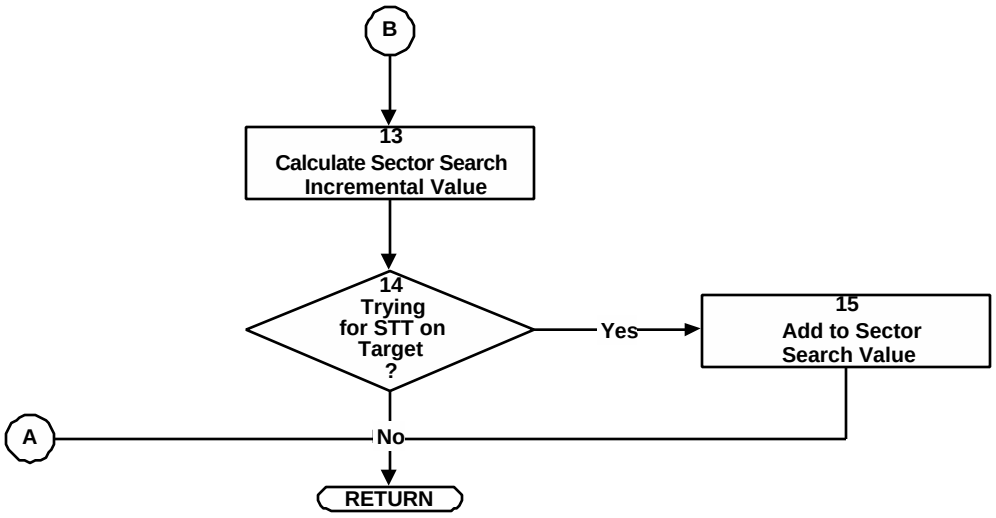

FIGURE 2.15-7. RDRVAL Functional Flow Diagram (Page 2 of 2).

Subroutine SCTSCH

Subroutine *sctsch* makes search sector decisions and conducts sector searches. Figure 2.15-8 is the functional flow diagram that describes the logic used to implement *sctsch*. The blocks are numbered for ease of reference in the following discussion.

Block 1. The end of search flag is set to FALSE. This flag will later be set to TRUE if either the search time is exhausted or there are no more sectors remaining to be searched.

Block 2. Test if the time available to search is less than 0.1 second. If true, go to Block 3. If false, jump to Block 4.

Block 3. Set the end of search flag equal to TRUE. Return control to calling routine (*sensor*) which will terminate the loop, ending the pilot's search of avionics and visual sectors.

Block 4. Calculate the remaining time available for the pilot to search the avionics and visual sectors.

Block 5. Test if the visual mode flag, hrlmode(1) is set. If true, go to Block 6. If false, jump to Block 7.

Block 6. Reset the time available to search equal to 0.5 second.

Block 7. Subroutine *bstsec* is called to determine which sector should be searched. It returns the sector number of the sector with the highest sector search value (previously determined in routines *setsvl* and, for the radar sector case, *rdrval*).

Block 8. Test if any sectors remain to be searched. If no sectors remain, go to Block 9. If there are sectors left, jump to Block 10.

Block 9. Set the end of search flag equal to TRUE. Return.

Block 10. Initialize the time used for the search to be the smaller of 1 second or the time available for searching. If this is the first consciousness event for this pilot, set the time of the observation to the current time. Otherwise, set the time at which the observation is made to the time at the start of the search plus one half of the time used.

Block 11. Test if the highest scoring search sector is a visual sector. If true, go to Block 12. If false, go to Block 13.

Block 12. Subroutine *vissch* is called to perform a visual search of a single sector and determine if what observations are made and how much time is used to make them.

Block 13. Test if the highest scoring search sector is the radar sector. If true, go to Block 14. If not, go to Block 15.

Block 14. Subroutine *rdrsch* is called to perform a search of the radar sector and determine what observations are made and how much time is used to make them.

Block 15. Test if the highest scoring search sector is an infrared search and track (IRST) sector. If true, go to Block 16. If false, go to Block 17.

Block 16. Subroutine *irssch* is called to perform a search of the IRST sector and determine what observations are made and how much time is used to make them.

Block 17. Test if the highest scoring search sector is an identify friend or foe (IFF) sector. If true, go to Block 19. If false, go to Block 19.

Block 18. Subroutine *iffsch* is called to perform a search of the IFF sector and determine what observations are made and how much time is used to make them.

Block 19. Test if the highest scoring search sector is a missile warning (MW) sector. If true, go to Block 20. If false, go to Block 21.

Block 20. Subroutine *mwsch* is called to perform a search of the MW sector and determine what observations are made and how much time is used to make them.

Block 21. Test if the highest scoring search sector is a missile approach warning (MAW) sector. If true, go to Block 22. If false, go to Block 23.

Block 22. Subroutine *mawsch* is called to perform a search of the MAW sector and determine what observations are made and how much time is used to make them.

Block 23. Decision whether the highest scoring search sector is a radar warning receiver (RWR) sector. If true, go to Block 24. If false, go to Block 25.

Block 24. Subroutine *rwrsch* is called to perform a search of the RWR sector and determine what observations are made and how much time is used to make them.

Block 25. Decision whether the highest scoring search sector is an integrated display (IND) sector. If true, go to Block 26. If false, go to Block 27.

Block 26. Subroutine *indsch* is called to perform a search of the IND sector and determine what observations are made and how much time is used to make them.

The ESM device is an enhancement that will be available in Version V6.3. For version V6.2, return at this point. For V6.3, continue with Block 27.

Block 27. Decision whether the highest scoring search sector is an electronic support measures (ESM) sector. If true, subroutine *esmsch* is called to perform a search of the ESM sector.

Block 28. Subroutine *esmsch* is called to perform a search of the ESM sector and determine what observations are made and how much time is used to make them. Then return.

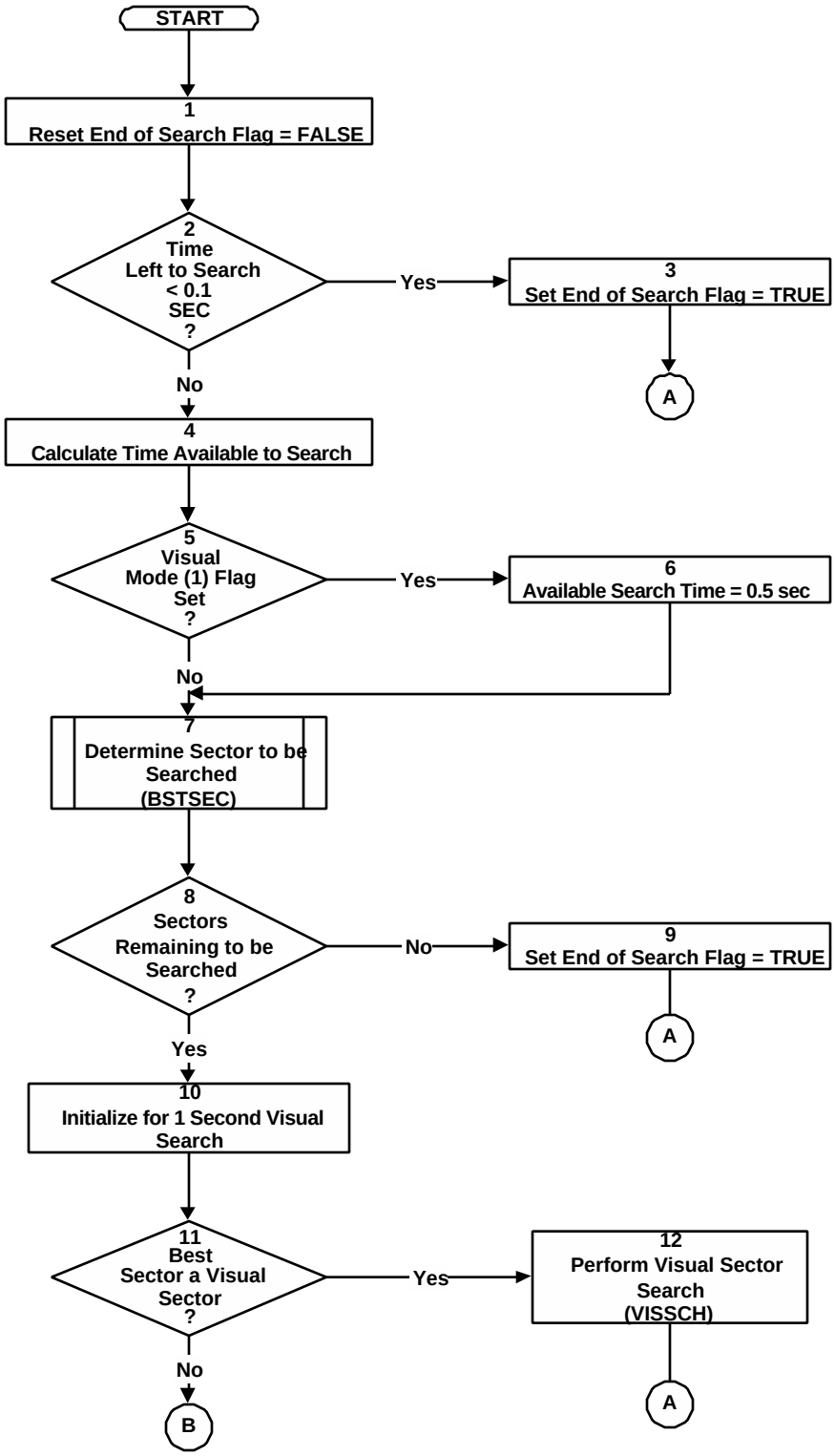

FIGURE 2.15-8. SCTSCH Functional Flow Diagram (Page 1 of 3).

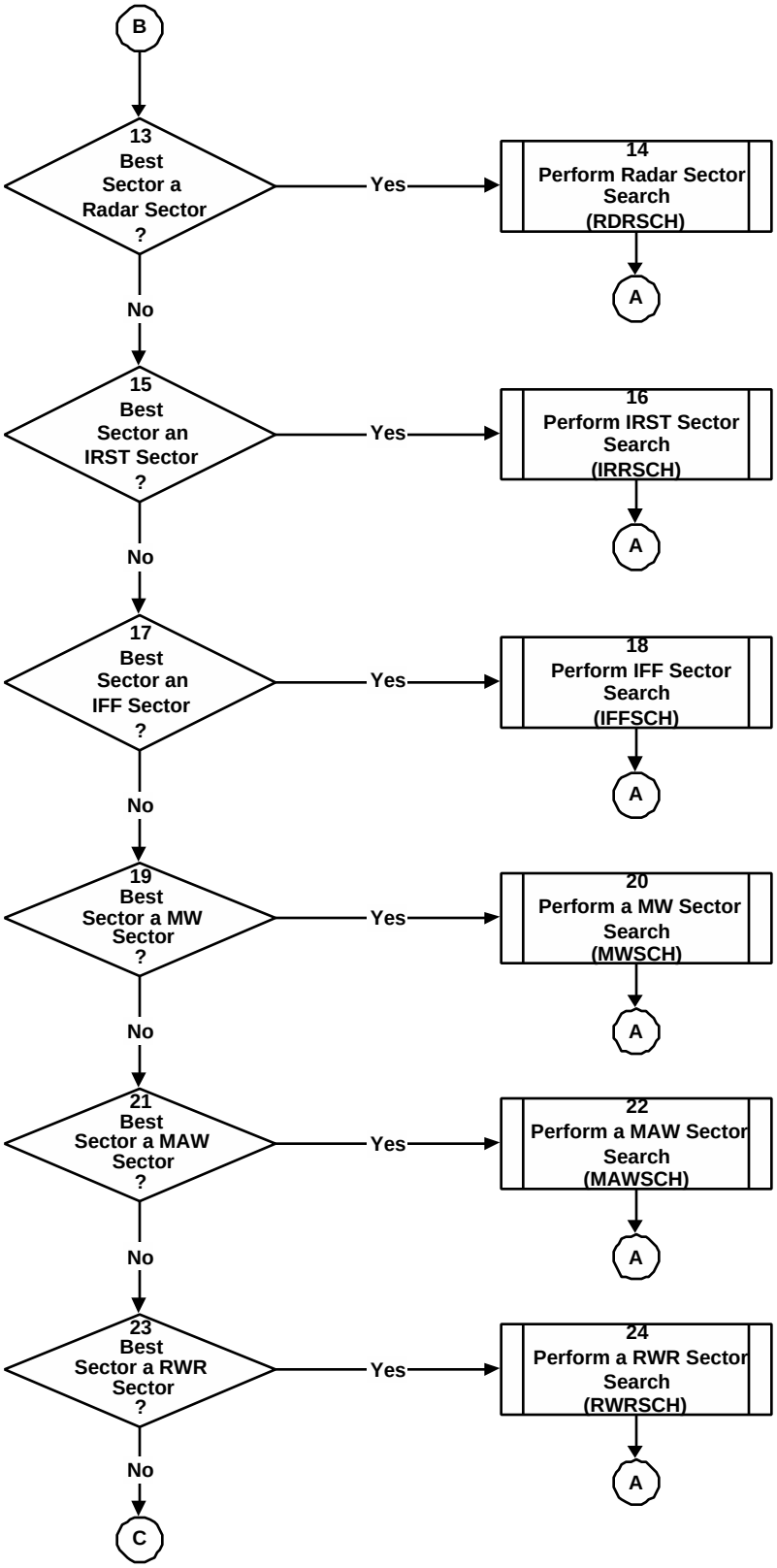

FIGURE 2.15-8. SCTSCH Functional Flow Diagram (Page 2 of 3).

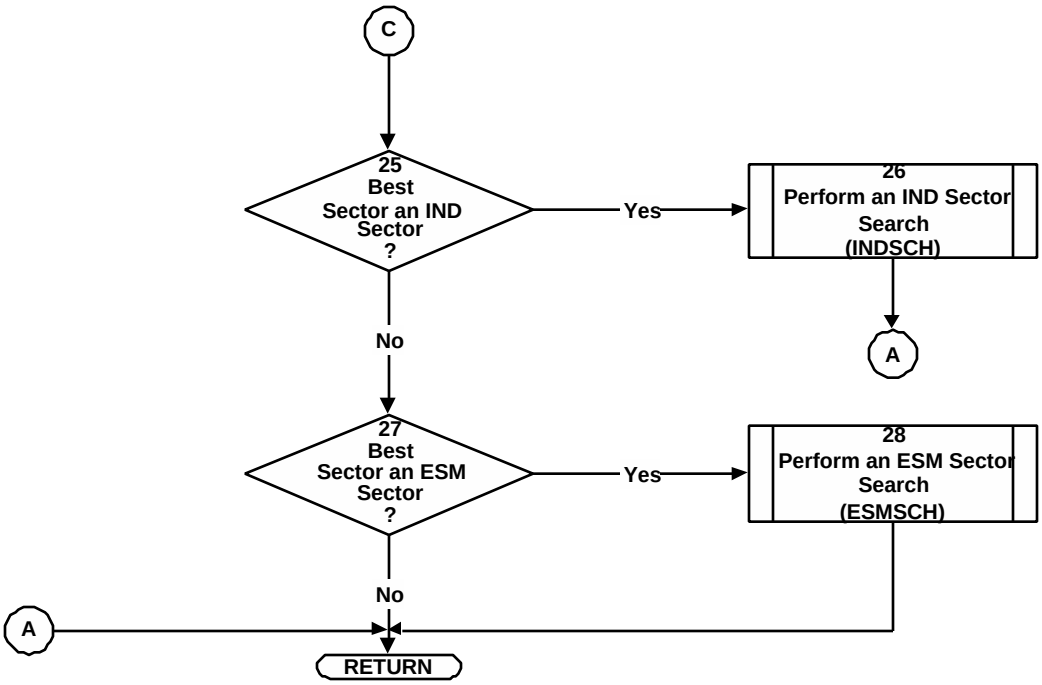

FIGURE 2.15-8. SCTSCH Functional Flow Diagram (Page 3 of 3).

Subroutine OBRDSC

Subroutine *obrdsc* produces a list of aircraft and missiles that the pilot will to observe when he looks at his radar scope. Figure 2.15-9 is the functional flow diagram that describes the logic used to implement *obrdsc*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Initialize time spent looking at the radar screen to be one second. This time will later be incremented by 0.1 second for each track displayed on the radar scope (Block 14).

Blocks 2-17 are in a loop over the maximum number of tracks that the radar trackbank may contain.

Block 2. Test if the current track is a valid track. If false, increment the track counter and go back to the top of the loop. Otherwise, continue processing the current track.

Block 3. Subroutine *rdrgtt* is called to obtain the radar track data from memory.

Block 4. Test if range information is available for this track. If not, go to Block 5. If so, go to Block 6.

Block 5. The distance to the radar track is set equal to one. This is used when no range information is available.

Block 6. The distance to the radar track from the conscious pilot's aircraft position and the radar track position are calculated via a call to the function *dist*.

Block 7. Test if the distance to the radar track is greater than the scope limit. The scope limit is the maximum range that the radar scope is currently displaying targets. This is initialized to a value of 1000 nmi so as to always display all tracks, but can be reset through production rules. If the track is outside the limits, increment the track counter and proceed to the top of the loop (jump to Block 17). Otherwise, continue processing the current track.

Block 8. Subroutine *rrtklh* is called to determine the last radar hit time, mode and antenna number for this track.

Block 9. Test if the time of the last update (hit on radar track) is valid. This is a defensive check, since the track update time should always be a valid time. The ESA radar is run for a short time, given by the variable *iniint*, before the simulation starts in order to build up a reasonable set of tracks at t=0, so valid update times can range from *-iniint* to the current simulation time. For non ESA radars, valid update times can range from zero to the current update time. If the update time is not valid, go to Block 10. Otherwise, continue processing the current track.

Block 10. Subroutine *nabort* is called to terminate the run and print diagnostic messages due to an invalid track update time (see Block 9).

Block 11. Test if the track has disappeared from the radar scope due to maximum display time limits. If the radar mode is scan or single target track (STT) and the time since the last track update is greater than the duration time on the radar scope, then this track has faded from view. Skip it, increment the track counter and proceed to the top of the loop. Otherwise, continue processing the current track. The duration time on the radar scope is calculated as the number of frames stored on the display times the nominal frame time. The number of frames stored is a characteristic of the radar and is read in during initialization.

Block 12. Test if the track is established and the radar is in track while scan (TWS) mode or SPOT mode. If not, then increment the track counter and proceed to the top of the loop. Otherwise, continue processing the current track.

Block 13. Test if the track is established and the radar is an ESA radar. If not established, then increment the track counter and proceed to the top of the loop. Otherwise, continue processing the current track.

Block 14. The track has passed cuts and will be displayed on the scope. Increment the search time by 0.1 second for the pilot to look at this track on the radar scope.

Block 15. Test if the track has been updated since the last radar search. If it has, go to Block 16. Otherwise, the track hasn't changed, so there is no new information to be gained by generating and processing an observation. Increment the track counter and go back to the top of the loop.

Block 16. Subroutine *mklosb* is called to add this track to the cumulative list of entities that will be observed.

Block 17. Test if there are any tracks are remaining. If true, increment the track counter and proceed to the top of the loop.

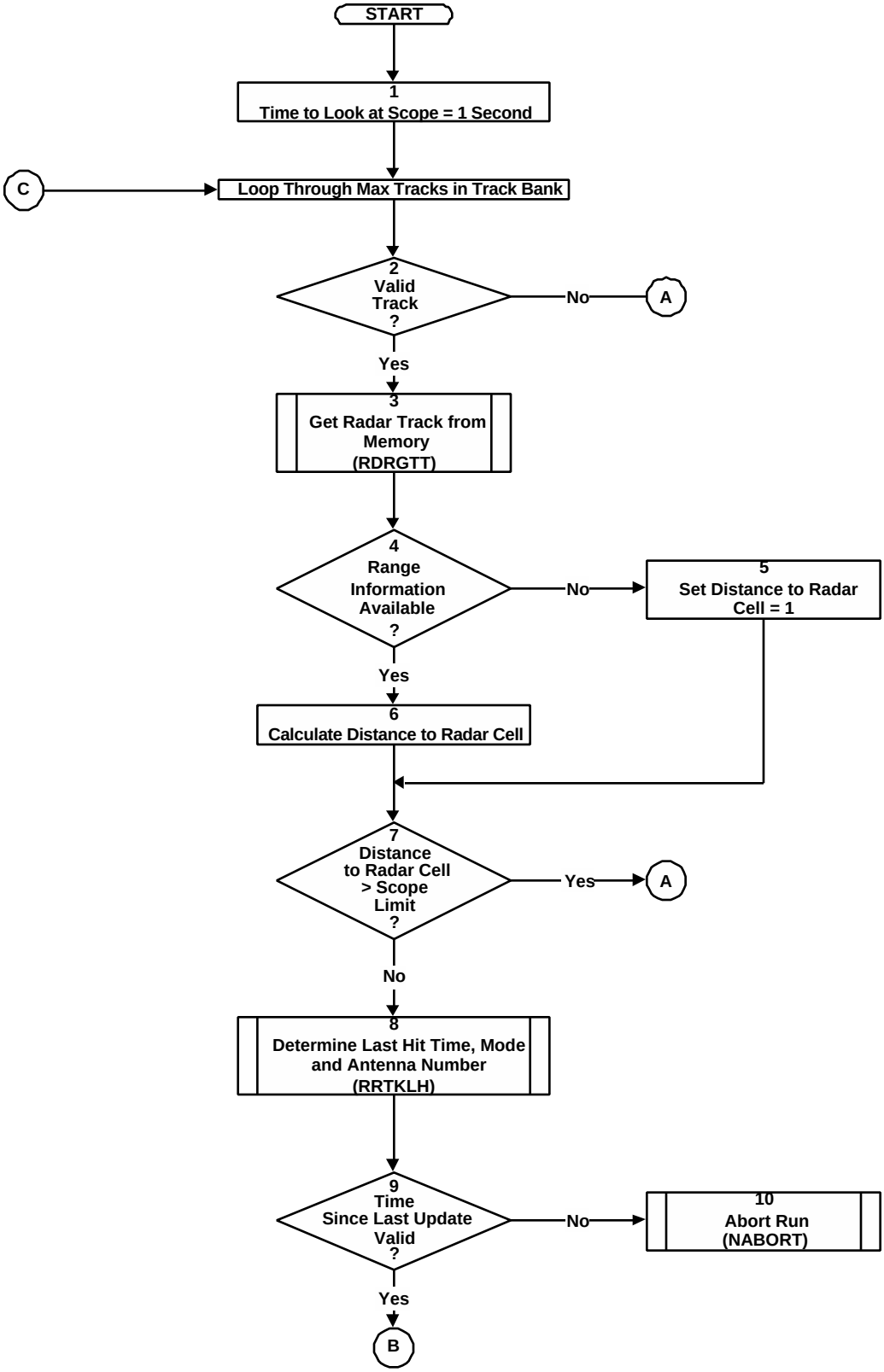

FIGURE 2.15-9. OBRDSC Functional Flow Diagram (Page 1 of 2).

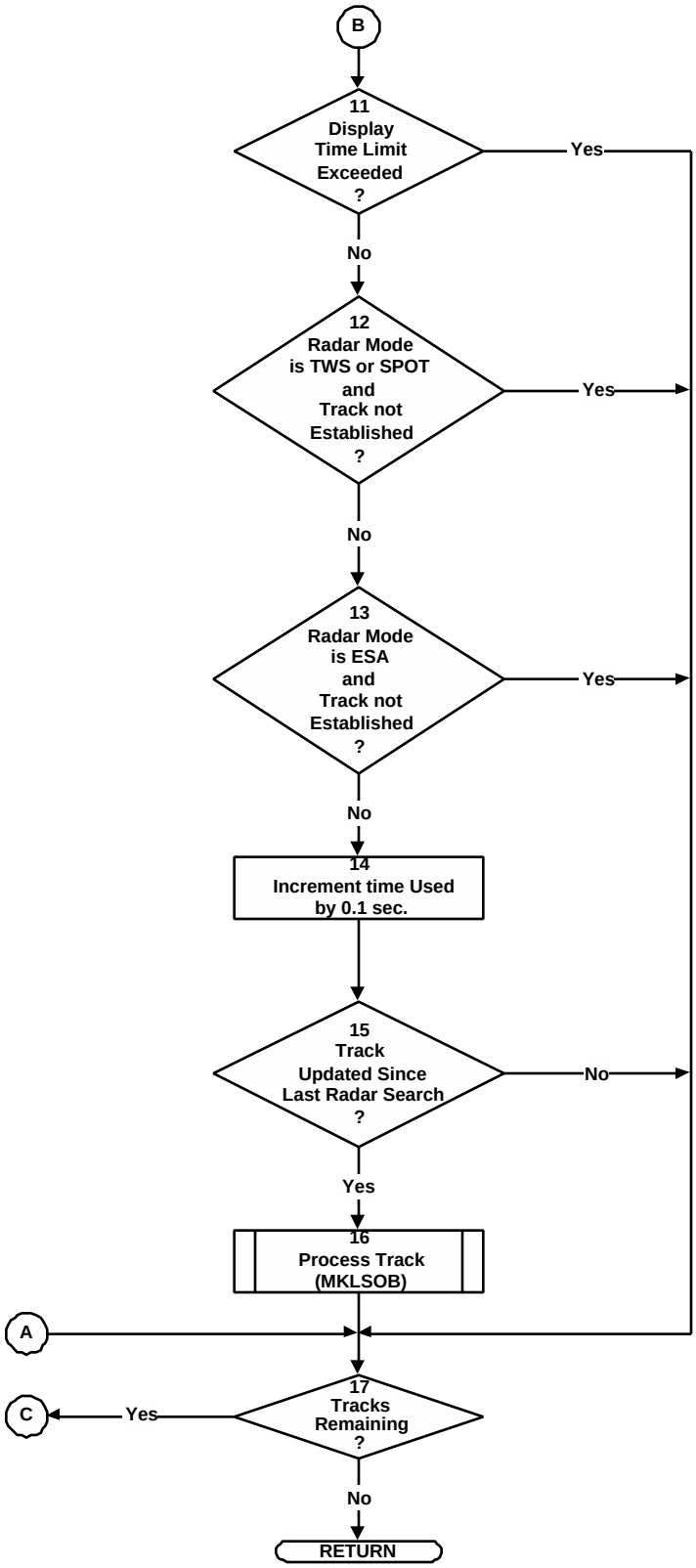

FIGURE 2.15-9. OBRDSC Functional Flow Diagram (Page 2 of 2).

Subroutine RRSNSD

Subroutine *rrsnsd* generates radar observations for the pilot. Figure 2.15-10 is the functional flow diagram that describes the logic used to implement *rrsnsd*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Test if the number of observed entities is zero. If true, return to calling routine, otherwise continue processing the observation data.

Blocks 2-23 are in a loop over the number of observed entities.

Block 2. Subroutine *rdrgtt* is called to obtain the radar track data from memory for the observed aircraft.

Block 3. Subroutine *prjacc* is called to make a constant acceleration projection of the observed track's position and velocity from the last update time to the current simulation time.

Block 4. Subroutine *vecinc* is called to calculate an incremental position for the conscious pilot's aircraft. This is done so that the later call to *rdrbem* will not put the conscious pilot's own aircraft on the list of observed targets.

Block 5. Subroutine *vsub* is called to subtract the conscious pilot's incremental position (Block 4) from the target position to generate a line of sight along the radar beam.

Block 6. Subroutine *rdrbem* is called to make a list of all aircraft within a radar beam and a list of stand off jammers (SOJ) that are jamming into the beam. The starting point for the beam is the incremental position computed in Block 4, and the direction is the line of sight computed in Block 5. The beam width in azimuth and elevation are characteristics of the current mode of the radar, and are read in during initialization.

Block 7. Test if the observed entity is an aircraft. If true, process the observation for an aircraft (Blocks 8-15). Otherwise, assume the entity is a missile and process the observation for a missile (Blocks 16-22).

Blocks 8-15 are executed when the observed entity is an aircraft.

Block 8. Test if the observed aircraft is in a track that is being jammed (either by aircraft in the track or by a SOJ). If true, go to block 9. If false, go to Block 10.

Block 9. Set the detection method = 5 (jammed radar).

Block 10. Set the detection method = 2 (clear radar).

Block 11. Subroutine *radobs* is called to compute the observables for this observation (this routine is detailed below). The time of the radar track and the target tail number are also recorded.

Block 12. Test if the observed aircraft is attempting to jam and the submodel flag 'ouemod(4)' is set. This flag allows the pilot to type an aircraft if it is attempting to jam his radar, but he still detects it. If true, go to Block 13. Otherwise, go to Block 14.

Block 13. Typing of observed aircraft is allowed. Record the aircraft type and the fact that the typing was accomplished via electronic means as part of the observation data.

Block 14. Typing of observed aircraft is not made. Record that no type information was obtained and that the quality of the observation is "bvr_id_md", indicating that no identifying information is contained in this observation.

Block 15. The aircraft is recorded as being observed as alive.

Blocks 16-22 are executed when the observed entity is a missile.

Block 16. Subroutine *mxref* is called to get the missile slot number from the missile identification number.

Block 17. Test if the missile slot number contains a valid missile. If not, process the next observed entity (Block 18). Otherwise, continue to Block 19.

Block 18. Increment the loop counter and proceed to the top of the loop to process the next observed entity.

Block 19. Test if the observed missile is in a jammed track. If true, go to Block 20. If false, go to Block 21.

Block 20. Set the detection method = 5 (jammed radar detection).

Block 21. Set the detection method = 2 (clear radar).

Block 22. Subroutine *radobs* is called to compute the observables for this observation. The time, missile ID number, and fact that the missile is still "alive" (not exploded) are also recorded as part of the observation data.

Block 23. Test if there are any more observed entities to be processed. If true, increment the loop counter and proceed to the top of the loop to process the next observed entity. Otherwise, return control to the calling routine.

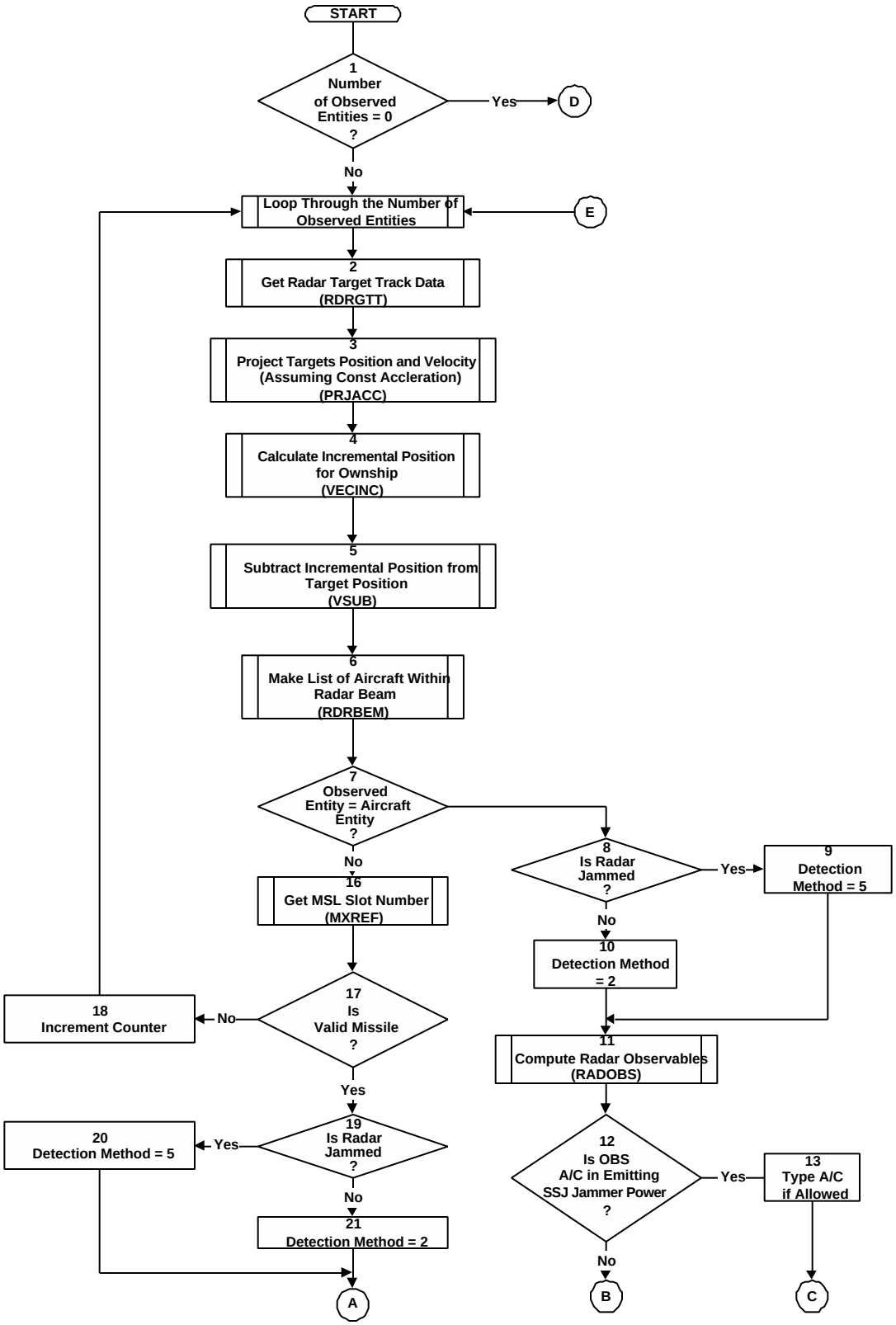

FIGURE 2.15-10. RRSNSD Functional Flow Diagram (Page 1 of 2).

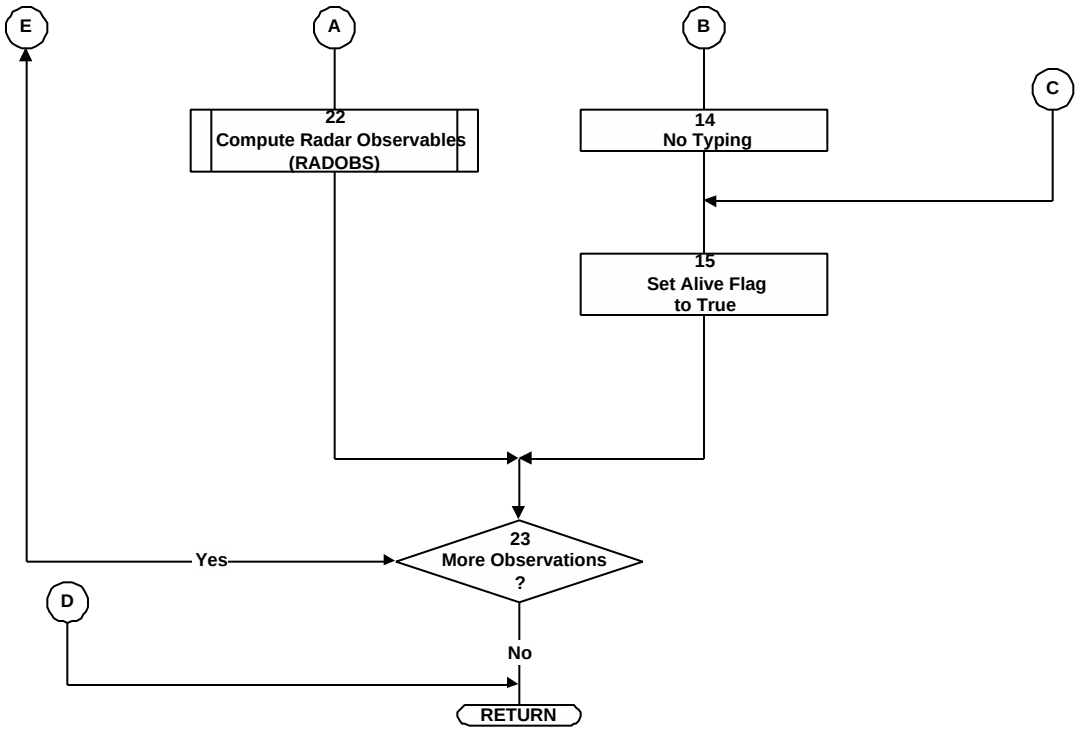

FIGURE 2.15-10. RRSNSD Functional Flow Diagram (Page 2 of 2).

Subroutine RADOBS

Subroutine *radobs* computes radar observables, errors, and cross correlation matrices. Figure 2.15-11 is the functional flow diagram that describes the logic used to implement *radobs*. The blocks are numbered for ease of reference in the following discussion.

Block 1. If the perfect information flag is set to true, go to Block 2. Otherwise, go to Block 3.

Block 2. The observation pointer is set to null, indicating that no observation is generated, since perfect information is being used. An observation is not required in this mode because ground truth information will be used to update the pilot's mental model. After this, return to the calling routine.

Block 3. Subroutine *rdrgtt* is called to obtain the radar track data from memory for the track to be observed.

Block 4. Subroutine *rrtklh* is called to determine the last radar hit time, mode and antenna number for this track.

Block 5. Test if the mode of the last radar hit is 'scan' mode. If true, Blocks 9-16 are executed. Otherwise, Blocks 6-8 are executed.

Block 6. Set the observation type = 22, (earth centered Cartesian, position and velocity).

Block 7. Put the track position and velocity values into the observation.

Block 8. Copy the track cross correlation matrix into the observation cross correlation matrix. Continue at Block 17.

Block 9. Test if the latest observation type is “azel” (azimuth and elevation only). If true, go to Block 10. If false, go to Block 11.

Block 10. Populate the cross correlation matrix as a 2x2 matrix with diagonal elements of 1 degree uncertainties in azimuth and elevation. Continue at Block 16.

Block 11. Test if the latest observation type is “raerd” (azimuth, elevation, range and range rate data). If true, go to Block 12. Otherwise, go to Block 13.

Block 12. Populate the cross correlation matrix as a 4x4 matrix with diagonal elements of 1 degree uncertainty in azimuth and elevation, 500 ft uncertainty in range and 5 ft/sec uncertainty in range rate. Continue at Block 16.

Block 13. Test if the latest observation type is “azelrd” (azimuth, elevation and range rate data). If true, go to Block 15. If false, go to Block 14.

Block 14. An undefined observation type has occurred, the run is aborted and diagnostic messages are printed.

Block 15. Populate the cross correlation matrix as a 3x3 matrix with diagonal elements of 1 degree uncertainty in azimuth and elevation and 5 ft/sec uncertainty in range rate.

Block 16. Subroutine *oberr* is called to add the errors to the observation. The errors are constructed by generating a Gaussian random variate with a mean of zero and a variance of 1 and multiplying this by the square root of the diagonal element of the cross correlation matrix that corresponds to the observable. This is added to the observable to generated an errored observation.

Block 17. Subroutine *olistv* is called to store this observation in list memory and return a pointer to it.

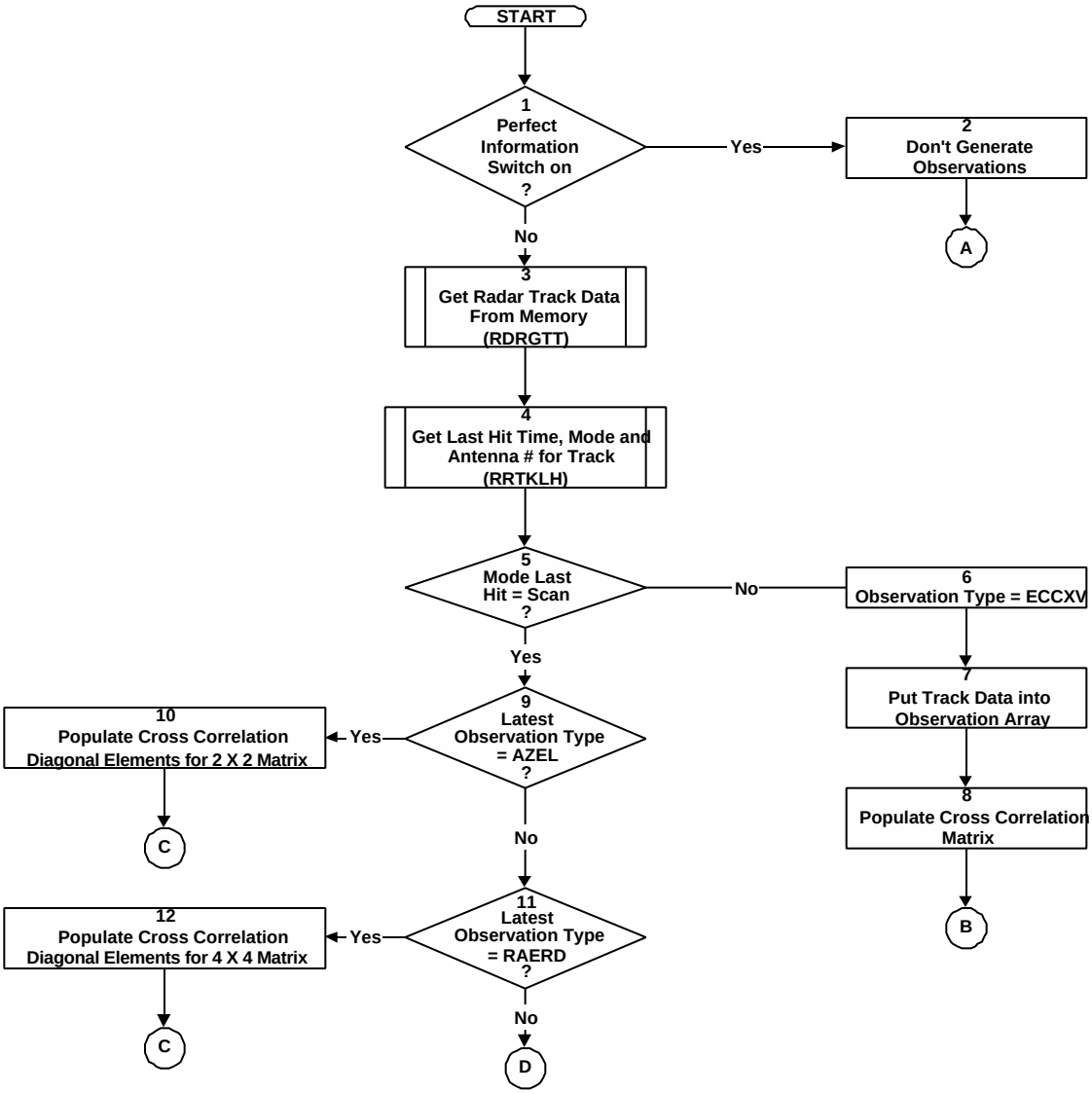

FIGURE 2.15-11. RADOBS Functional Flow Diagram (Page 1 of 2).

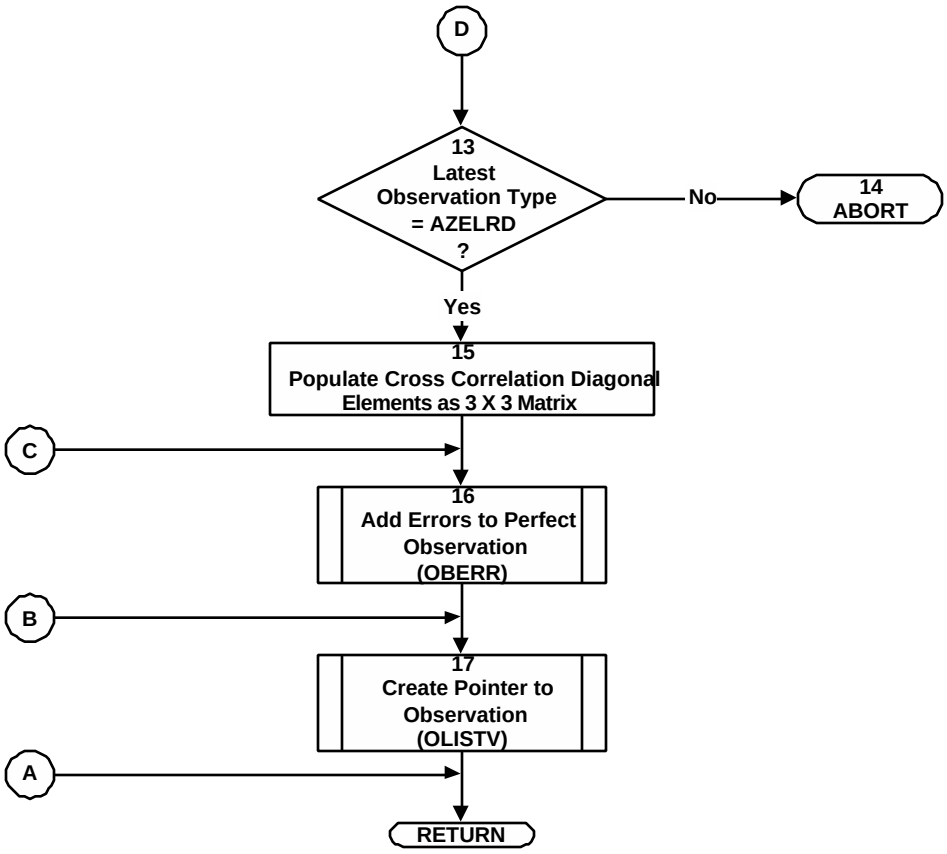

FIGURE 2.15-11. RADOBS Functional Flow Diagram (Page 2 of 2).

Subroutine MINDUP

Subroutine *mindup* is the executive routine for updating the mental model of the currently conscious pilot. Figure 2.15-12 is the functional flow diagram that describes the logic used to implement *mindup*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Update the radar status and characteristics data for this aircraft by calling subroutines *grdrs* and *grdrc*.

Block 2. Test if this consciousness event was triggered by the receipt of a radio message. If true, get messages about other aircraft and missiles (Blocks 3-5). Otherwise, skip to Block 6.

Block 3. Subroutine *inicmm* is called to set up for message receipts.

Block 4. Subroutine *rcvobs* is called to receive messages about other aircraft.

Block 5. Subroutine *rcvmob* is called to receive messages about missiles.

Block 6. Subroutine *ccself* is called to update the state vector information for the conscious pilot's own aircraft. This is basically a transfer of information from the common block that stores the external status of all aircraft (*extst*) into the pilot's mental model.

Block 7. Subroutine *msl2x0* is called to process all missiles for which observation information is present. This routine is analogous to *cc2x0* which is detailed below.

Block 8. Subroutine *cc2x0* is called to process all aircraft for which observation information is present. This routine is detailed below.

Block 9. Test if this consciousness event is a communications receipt event. If true, go to Block 10. If not, go to Block 11.

Block 10. Subroutine *rcv_intent* is called to process any intent to shoot messages. If another pilot is going to shoot at the currently conscious pilot's primary target, a consideration to changing targets is made by requesting a pilot posture decision, which will force a call to select a weapon/target pair (*selwpn*).

Block 11. Subroutine *astyp* is called in an attempt to determine the 'type' (hostile/friendly) of an aircraft based on the knowledge of the types of other members of that flight.

Block 12. Subroutine *asown* is called to attempt to determine the ownership of the missiles that the pilot knows about, if possible. If detected early enough, the missile may be linked to the firing aircraft by taking into account the geometry and proximity of other entities.

Block 13. Subroutine *astgt* is called to attempt to determine who the targets are of all of the missiles that the pilot knows about. The pilot already knows the targets of his own missiles, and he will received radio messages containing this information for missiles fired by his flightmates. For other missiles, he will attempt to project current positions and velocities to try to assess each missile's most likely target.

Block 14. Test the value of a user set mode flag (*ouemod(6)*). If true, go to Block 15. If not, go to Block 16.

Block 15. Subroutine *astyp2* is called to allow typing of an aircraft as hostile if it can be determined that it has fired on a friendly aircraft.

Block 16. Test if this consciousness event is a preplanted mission change event. If true, go to Block 17. If not, jump to Block 22.

Block 17. Process preplanted events (i.e. read new orders).

Block 18. Check event stack and determine if preplanted events need replanting at a future time. If so, reschedule the event by calling subroutine *plants*.

Block 19. Test if the mission has changed. If true, go to Block 20. If not, go to Block 21.

Block 20. Set flag to initiate a flight posture level decision. This is done if the mission has changed. This is the highest level in the decision hierarchy and determines the general course of action for the flight. Continue at Block 24.

Block 21. Set flag to initiate a flight tactics level decision, which is the second highest level decision. Continue at Block 24.

Block 22. Test if this consciousness event is a communications receipt event. If true, process orders received from the flight leader or GCI (Block 23). If not, go to Block 24.

Block 23. Subroutine *rcv_orders* is called to process any incoming orders from GCI/AWACS controllers or the flight leader.

Block 24. Subroutine *mmordr* is called to reorganize the pilot's mental model if necessary. Reorganization constitutes sorting the highest valued aircraft to the top of the list of aircraft to be taken under detailed consideration

Block 25. Test if the aircraft is damaged. This flag can only be set through production rules. If true, go to Block 26. If not, go to Block 27.

Block 26. The aircraft has been determined to be damaged, therefore, set the aggressiveness factor to 0.5.

Block 27. Test if it is time for a major update of the pilot's mental model. If it is, go to Block 28. If not, go to Block 29. A major update will be called if the time since the last major update is greater than the maximum decision interval or if the major update request flag was set due to a change in mission (Block 20) or the pilot made an observation containing significant new information.

Block 28. Subroutine *majud* is called to perform a complete situation assessment update. Some items included in this assessment are:

Update physical relationship variables
Ask for new decisions at all levels
Assess the likelihood that each friendly has been detected by the hostiles
Calculate 1v1 self engagement measures
Calculate expected valued killed
Compute effective and situational values
Compile high utility attack and evade lists

Block 29. Subroutine *minud* is called to perform a partial situation assessment. *Minud* updates self engagement measures (SEM) for self versus hostiles and hostiles versus self only. It also recomputes the utility of engagements using new SEMs.

Block 30. Subroutines *preobs*, *premob*, and *prereq* are called to prepare outgoing messages for flightmates. *Preobs* prepares the aircraft observations made by the conscious pilot. *Premob* prepares the missile observations made by the conscious pilot. *Prereq* prepares a message requesting information about unobserved aircraft that the conscious pilot is sufficiently worried about to ask for an update on.

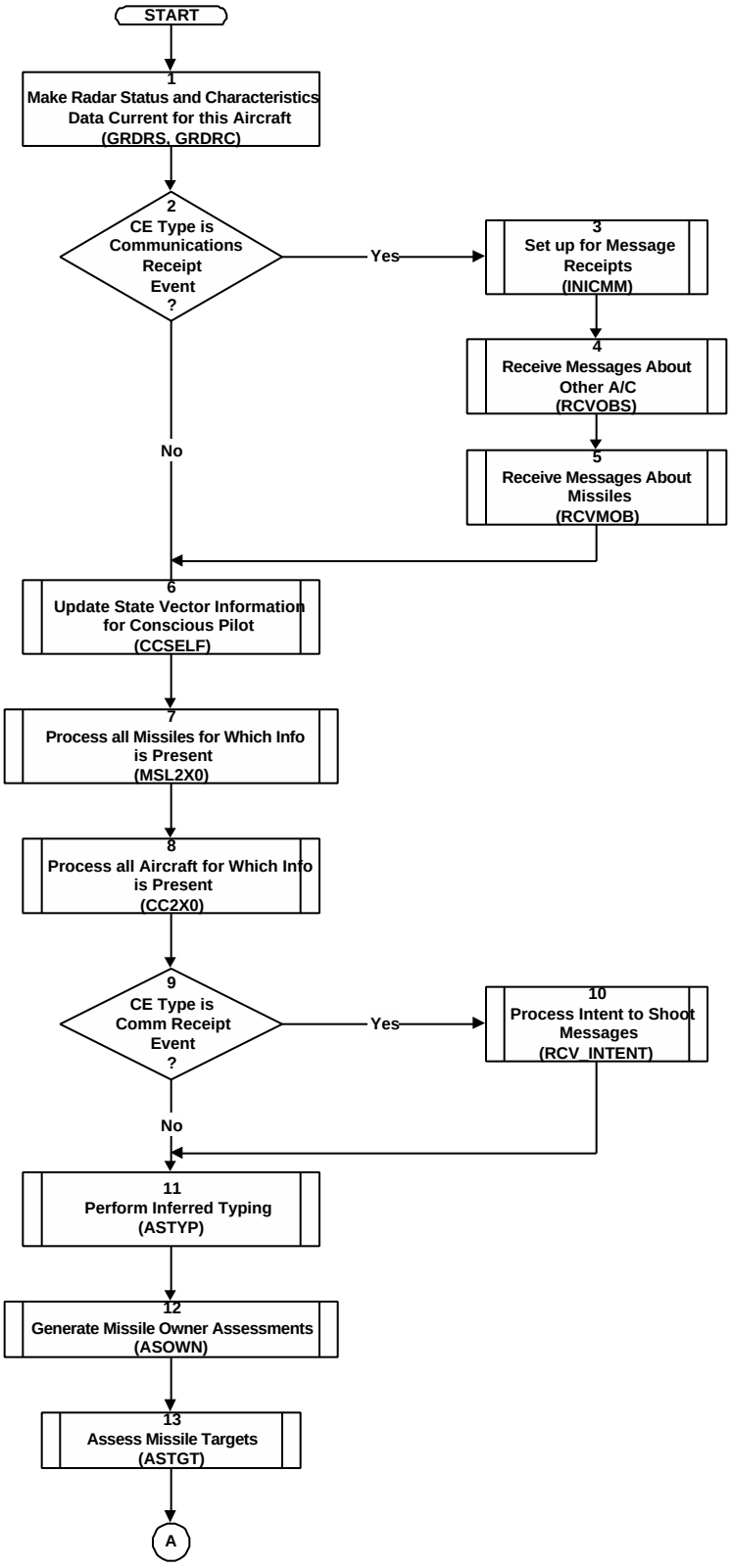

FIGURE 2.15-12. MINDUP Functional Flow Diagram (Page 1 of 3).

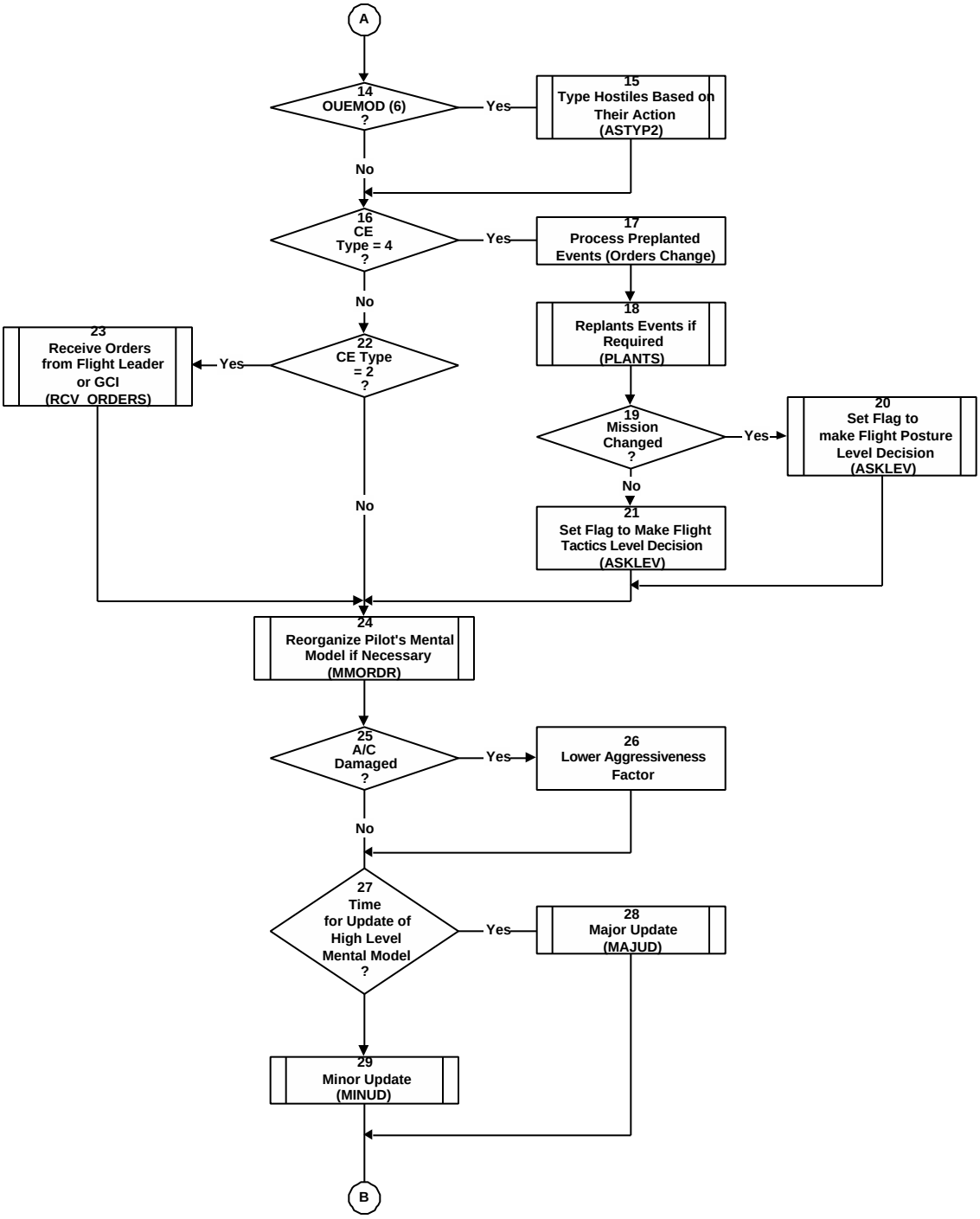

FIGURE 2.15-12. MINDUP Functional Flow Diagram (Page 2 of 3).

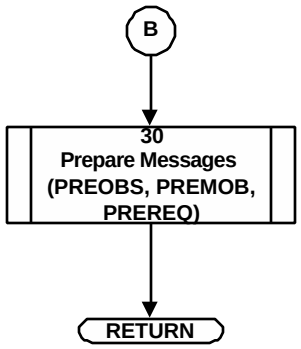

FIGURE 2.15-12. MINDUP Functional Flow Diagram (Page 3 of 3).

Subroutine CC2X0

Subroutine *cc2x0* is the top level routine for merging aircraft observations into the pilot's mental model. Figure 2.15-13 is the functional flow diagram that describes the logic used to implement *cc2x0*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Initialize variables for diagnostic information, the altered aircraft list, number of, observations with positional information. Also, save the IDs and relationships of all of the aircraft known to the pilot before this routine executes. This will be compared with the same information after observations have been processed in order to help determine if a significant change has occurred in the pilot's situation awareness.

Blocks 2-29 are in a loop over the number of observations.

Block 2. Subroutine *cc2x1* is called to correlate the observation with targets in the mental model to determine if the observation is worth processing (Subroutine *cc2x1* is detailed below).

Block 3. Test if the observation is worth processing. If not, then jump to Block 27. Otherwise, continue.

Block 4. Test if the pilot can determine that the aircraft under consideration is dead. The pilot will be able to infer this if the dead aircraft has been tracked for more than 5 seconds after it is killed. He will also know that it is dead if the perfect information switch is set to true and the aircraft is on the ground. If determined to be dead, then set the aircraft alive flag to false (Block 5). Otherwise, set it to the value of the variable daaliv, which is part of the observation data.

Block 5. The aircraft alive flag is set to false because the pilot was able to determine that the aircraft under consideration is dead.

Block 6. Test if this is the first observation of this aircraft. If true, go to Block 7. If not, jump to Block 13.

Block 7. Test if the aircraft is dead or was previously observed as dead. If true, then delete the observation pointer from memory (jump to Block 27), increment the loop counter and proceed to the top of the loop to process the next aircraft. If false, continue.

Block 8. Subroutine *cc2x3* is called to add this observation to the conscious pilot's mental model.

Block 9. Set the 'best information' variable for this aircraft. This records the most accurate or complete information source that is contributing to this track.

Block 10. Set flag to force a major update of the conscious pilot's mental model because a new detection is a significant piece of information.

Block 11. Subroutine *addmsg* is called to add the newly detected aircraft to the outgoing message list unless it is in the conscious pilot's own flight or is on the same mission or if this initial detection is itself the result of a radio message received from someone else.

Block 12. Subroutine *adlist* is called to add the aircraft to the altered aircraft list. This is a list of all aircraft in the pilot's mental model about which his knowledge has changed significantly during this consciousness event.

Blocks 13-26 are executed when the observed aircraft is already known to the conscious pilot.

Block 13. Test if the pilot can observe that the aircraft is dead. If true, execute blocks 14-16 to remove it from the conscious pilot's mental model. If false, execute blocks 17-25 to process the observation.

Block 14. Subroutine *addmsg* is called to inform others of the observed aircraft's death. A message will be added to the outgoing message list.

Block 15. Subroutine *mremac* is called to remove the observed aircraft from the conscious pilot's mental model and to add it to the list of aircraft that he believes to be dead.

Block 16. Subroutine *listd* is called to free the list memory space holding the observation. Jump to Block 27.

Block 17. Subroutine *cc2x2* is called to update the pilot's perception of the aircraft's state with the new observation.

Block 18. Test if the observed aircraft is newly identified and friendly. If true, go to Block 19. If not, skip to Block 20.

Block 19. Set flag to request a major mental model update and a new pilot posture decision (unidentified aircraft are assumed to be hostile, so whenever one is newly identified as friendly, the pilot needs to reconsider his weapon/target decisions to prevent possible fratricide).

Block 20. Reset the 'best information' switch for this aircraft. If the method of observation for this observation is better than the previous best method of observation, then replace the best method of observation with the method of observation for this track. This keeps track

of the best information source that is contributing to each of the tracks in the pilot's mental model.

Block 21. Test if new identification information resulted from an on board detection and the newly identified (not newly observed, e.g. may now have visual identification instead of radar identification) aircraft is not in my flight. If true, send an observation message (Block 22). If not, skip to Block 23.

Block 22. Subroutine *addmsg* is called to add a newly identified aircraft message to the outgoing message list.

Block 23. Test if a message request concerning the observed aircraft has been received from another friendly aircraft. If true, send an observation message (Block 24).

Block 24. Subroutine *addmsg* is called to add this aircraft to the list of outgoing messages, if it isn't already there.

Block 25. Test if a significant change to the pilot's knowledge of the observed aircraft has occurred. If true, go to Block 26. If not, jump to Block 27.

Block 26. Subroutine *adlist* is called to add the observed aircraft to the altered aircraft list.

Block 27. Test if the observation pointer is not zero. If true, go to Block 28. If not, go to Block 29.

Block 28. Subroutine *listd* is called to delete the observation from memory.

Block 29. Test if there are any more observations remaining to be processed. If true, go back to Block 2. Otherwise, continue at Block 30.

Block 30. Subroutine *cc2_gcirng* is called to check for range data from GCI controllers and to add this to the pilot's mental model.

Block 31. Subroutine *cc2x4* is called to update all aircraft in the mental model to the current time, whether they were observed or not.

Block 32. Zero any elements of the mental model cross reference arrays for aircraft that have been removed from the mental model.

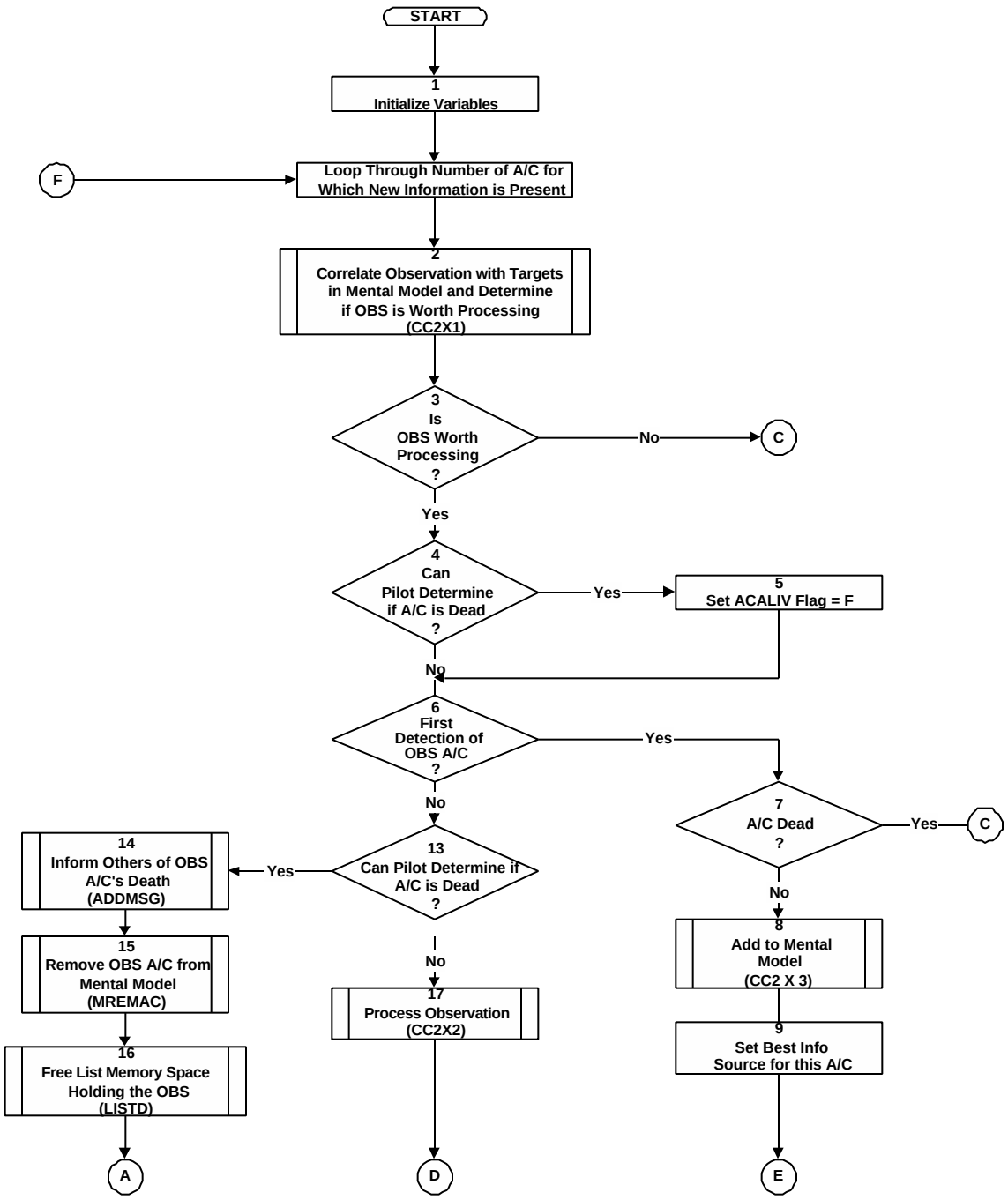

FIGURE 2.15-13. CC2X0 Functional Flow Diagram (Page 1 of 3).

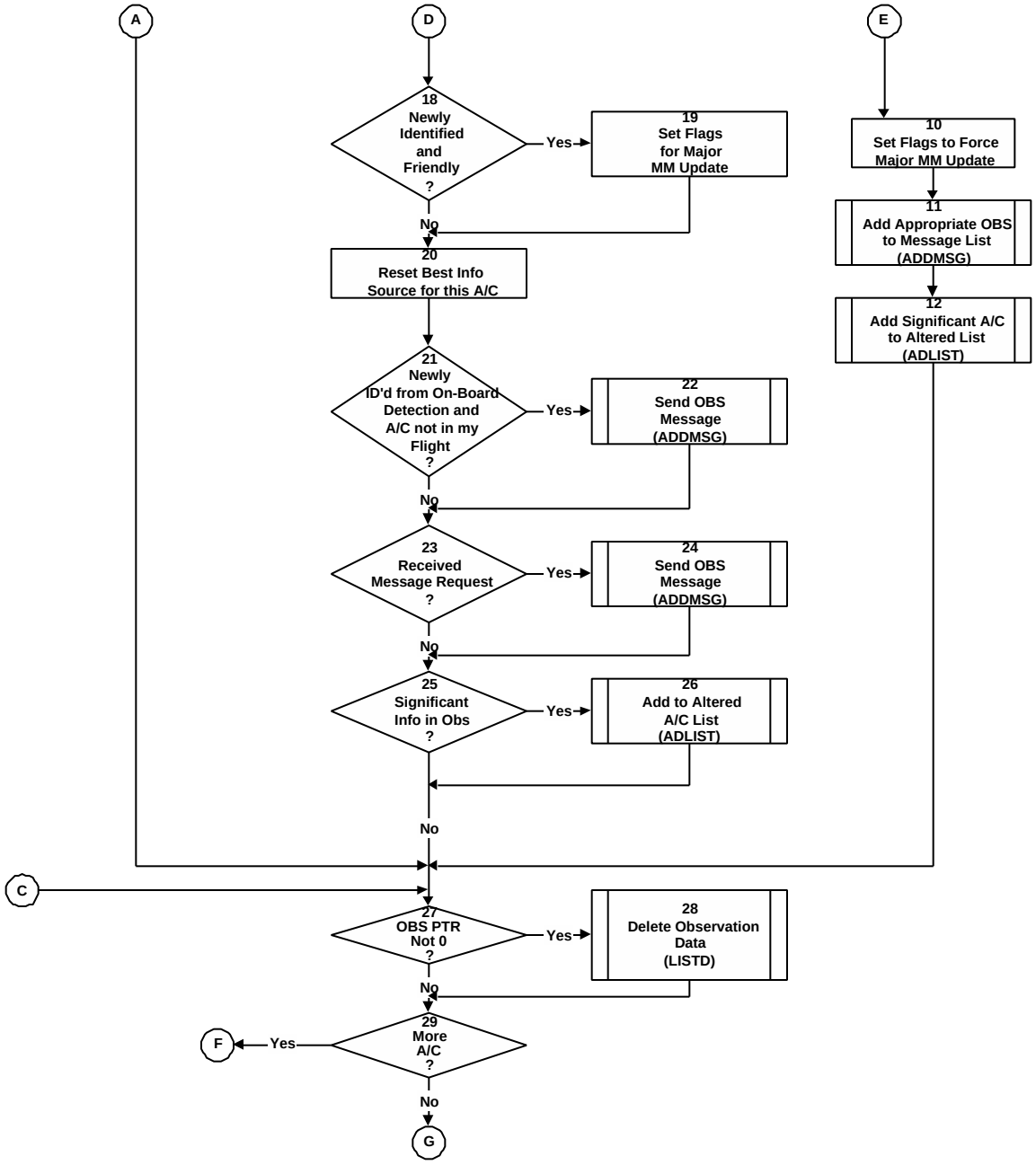

FIGURE 2.15-13. CC2X0 Functional Flow Diagram (Page 2 of 3).

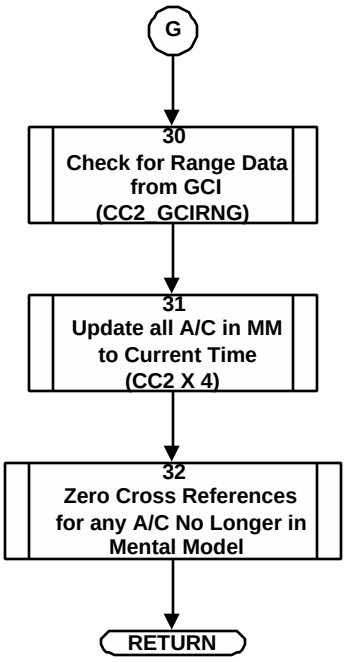

FIGURE 2.15-13. CC2X0 Functional Flow Diagram (Page 3 of 3).

Subroutine CC2X1

Subroutine *cc2x1* correlates observations with targets in the conscious pilot's mental model. It also determines whether or not it is worth continuing to process the observation under consideration. The observation may be discarded if it will not add any new information beyond what is already contained in other observations made during this consciousness event. *cc2x1* also records the reason for accepting or discarding an observation. The reasons can be printed out if the diagnostic print switch for this routine is turned on. Figure 2.15-14 is the functional flow diagram that describes the logic used to implement *cc2x1*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Subroutine *match* is called to determine the index of the observed aircraft in the conscious pilot's mental model (variable *ispotd*). This index is return as zero if the aircraft is not yet in the mental model (i.e. new observation).

Block 2. Test whether a limited information detection has been made. A detection is considered limited information if it is either an ECM, a jammed communications or an IRST detection. If the detection is not a limited detection, then accept the detection (Block 3) and return control to the calling routine. If the detection is a limited detection, go to Block 4.

Block 3. A normal observation was made, accept the detection. Set the return variable *discrd* to false, set the reason to "NORMAL OBSERVATION". Return to the calling routine.

Block 4. Subroutine *detinf* is called to check all of the observations made during this consciousness event and to return flags indicating what quality of observations have been made. The intent here is that if other observations already contain more information than the limited observation being examined here, there is no reason to continue to process this observation. *detinf* will return flags indicating whether visual, radar, message, ECM, non-ECM or inferred information is available.

Block 5. Test if visual information is available on the observed aircraft. If so, go to Block 6. If not, jump to Block 7.

Block 6. Discard this ECM/IRST detection since a visual detection was also made. Set the return variable *discrd* to true, the reason to "ACCOMPANYING VISUAL", and return.

Block 7. Test if the observation under consideration is an IRST detection. If true, continue at Block 8. If not, jump to Block 11.

Block 8. Test if radar information is also available on the observed aircraft. If true, discard this detection (IRST) in favor of the radar detection (Block 9). Otherwise accept the IRST detection (Block 10).

Block 9. Discard this detection (IRST) in favor of the radar detection. Set the return variable *discrd* to true, the reason to "IRST, also RADAR", and return.

Block 10. Accept this detection (IRST), since no better observations are available (visual or radar). Set the return variable *discrd* to false, the reason to "IRST detection", and return.

Block 11. Test if the observed aircraft is already known to the pilot. If true, continue at Block 12. If not, jump to Block 15.

Block 12. Test if the detection is a jammed message detection. If true, go to Block 13. Otherwise, the detection must be an ECM detection of a known aircraft, skip to Block 14.

Block 13. Discard this detection (jammed message about a known aircraft). Set the return variable *discrd* to true, the reason to "ECM COMM OF KNOWN AC", and return.

Block 14. Accept this detection (ECM detection of a known aircraft). Set the return variable *discrd* to false, the reason to "ALREADY KNOWN AC", and return.

Block 15. Test if the detection is a jammed message detection. If true, go to Block 16. Otherwise, go to Block 17.

Block 16. Accept this detection (a jammed message detection of an unknown aircraft). Set the return variable *discrd* to false, the reason to "ECM COMM OF UNKNOWN AC", and return.

Block 17. Accept this detection (ECM detection of an unknown aircraft). Set the return variable *discrd* to false, the reason to "ECM OF UNKNOWN AC", and return.

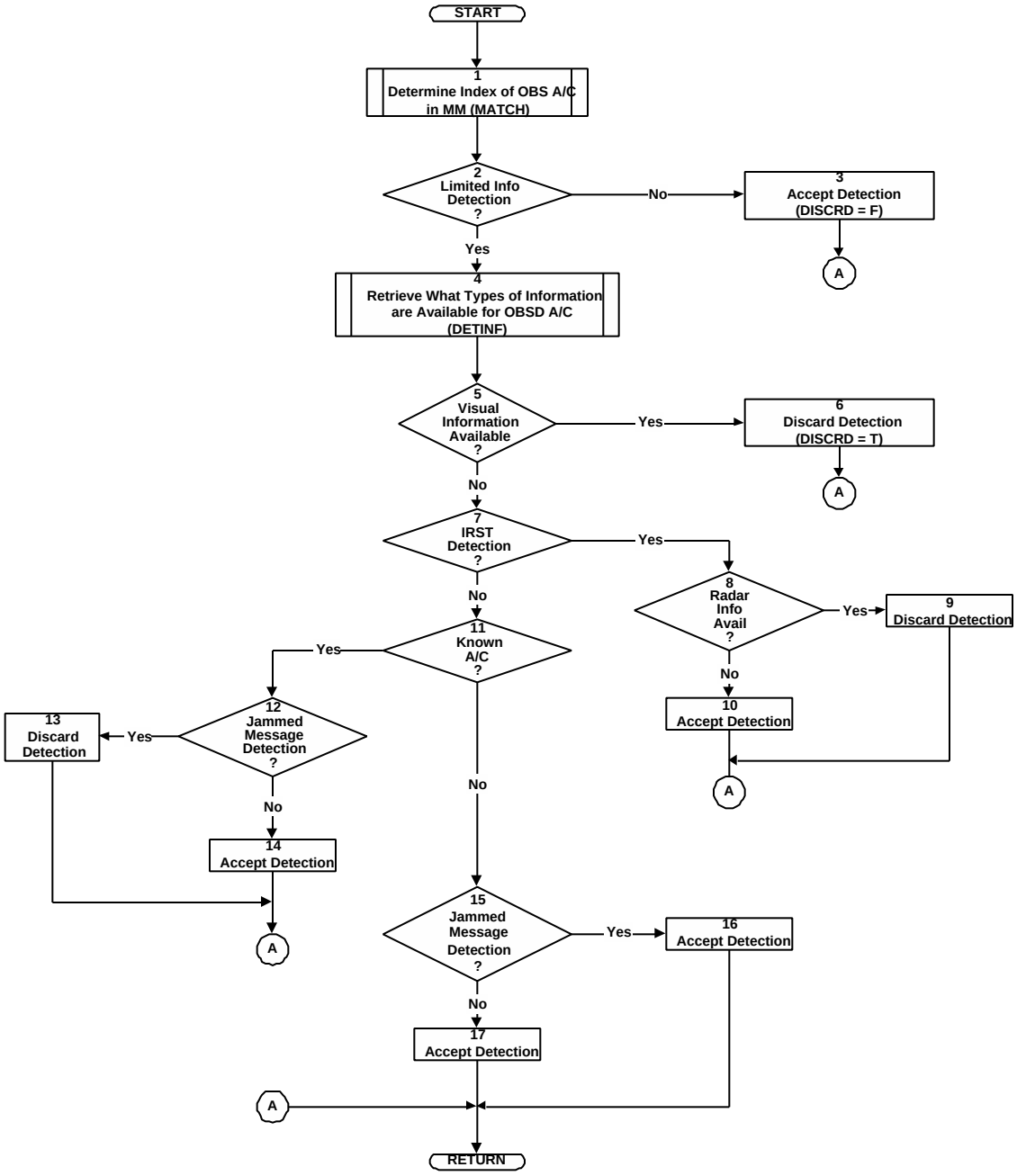

FIGURE 2.15-14. CC2X1 Functional Flow Diagram.

Subroutine CC2X3

Subroutine *cc2x3* adds a newly detected aircraft to the mental model. Figure 2.15-15 is the functional flow diagram that describes the logic used to implement *cc2x3*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Test if the HRL mode switch number 5 (*hrlmod*(5)) is set. If true, go to Block 2. If not, skip to Block 3.

Block 2. Subroutine *makecp* is called to take checkpoints after the number of detections has reached the predefined goal. The goal is currently set to be equal to four.

Block 3. If the number of known aircraft does not already equal or exceed the maximum number of aircraft in the pilot's detailed decision group (*mxacmm*), then the counter for the number of aircraft in the group (*nspotd*) is incremented. The counter for the total number of aircraft in the mental model (*ninmm*) is also incremented.

Block 4. The observed aircraft is added to the mental model cross reference arrays *iacidx*, *iacidt*, *mmindx*, and *mmindt*.

Block 5. Subroutine *prjacc* is called to project the position and velocity of the conscious pilot's aircraft to the time of the observation.

Block 6. Subroutine *prjacc* is called to project the position and velocity of the target aircraft to the time of the observation.

Block 7. The aircraft type and type quality are copied from the observation data to the pilot's mental model.

Block 8. If the aircraft is in the pilot's detailed consideration group, logical function *irhas* is called to determine if the detected aircraft poses an IR missile threat. *irhas* works as follows: if the type of the detected aircraft is unknown, IR capability is assumed to be true. If the type of the aircraft is known, IR capability is true if it is carrying any missiles with IR seekers, false otherwise.

Block 9. The newly detected aircraft's flight and element are recorded in the conscious pilot's mental model.

Block 10. If the newly detected aircraft is in the conscious pilot's detailed consideration group and is in his flight or element, record that information in his mental model.

Block 11. Subroutine *getrel* is called to assess the relationship of the new aircraft to the conscious pilot if the new aircraft is friendly. If the new aircraft is confirmed as hostile, the relationship is set to 'hostile but not currently engaged with self'. Otherwise, the relationship is set to unknown.

Block 12. The intrinsic value and combat effectiveness of the observed aircraft are recorded in the pilot's mental model. The same values are used whether the pilot knows the type of the aircraft or not.

Block 13. Subroutine *gtypd* is called fetch characteristics data for the observed aircraft.

Block 14. Record the method (sensor) used to make the detection.

Block 15. If the newly detected aircraft was visually sighted, record the time of first visual sighting.

Block 16. Record the observed afterburner state (on, off, unknown) of the newly detected aircraft.

Block 17. Subroutine *gtypd* is called to get information on external stores of the newly detected aircraft if they can be visually observed.

Block 18. Real function *dist* is called to calculate range from the conscious pilot's aircraft to the newly detected aircraft.

Block 19. Test if the perfect information flag is set. If set to true, go to Block 20. Otherwise, go to Block 22.

Block 20. Subroutine *m3perf* is called to use ground truth to update the pilot's knowledge of the observed aircraft's position, velocity, and acceleration.

Block 21. The mental model variables for time of observation (*tsight* = current simulation time), positional information available (*posinf* = true) and target state vector time (*svtimx* = current simulation time) are set.

Blocks 22-27 are executed if the perfect information flag is not set true.

Block 22. Test if positional information is available with this observation. If true, go to Block 24. If not, go to Block 23.

Block 23. Subroutine *nabort* is called to terminate the run and print diagnostic messages.

Block 24. Subroutine *inlsvd* is called to retrieve the observation data.

Block 25. Subroutine *tkupi* is called to initialize the state vector of the tracker for the first detection. Inputs are taken from the observation data. Mental model values for aircraft position, velocity and acceleration are then taken from the newly created state vector.

Block 26. Test if ground controller intercept (GCI) information is available and is not too old. If true, add range information to the conscious pilot's mental model (Block 27).

Block 27. Subroutine *gci_r_info* is called to add range information to the conscious pilot's mental model. This information is accepted only if it offers an improvement in the information already taken from the observation data. If accepted, a Kalman filter is employed to fold in the new range data to the mental model's state vector.

Block 28. Initialization of miscellaneous mind variables is completed before returning control to the calling routine.

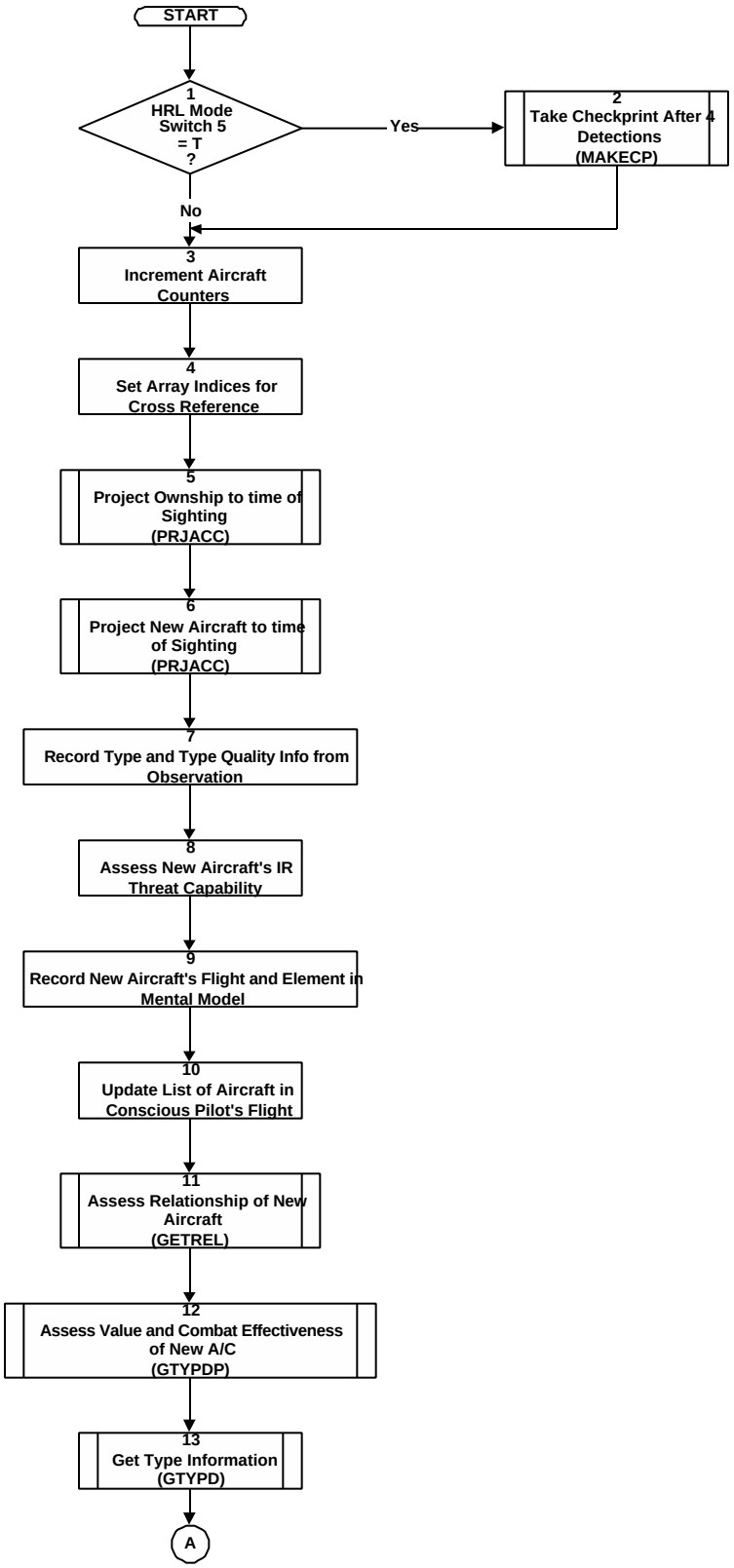

FIGURE 2.15-15. CC2X3 Functional Flow Diagram (Page 1 of 2).

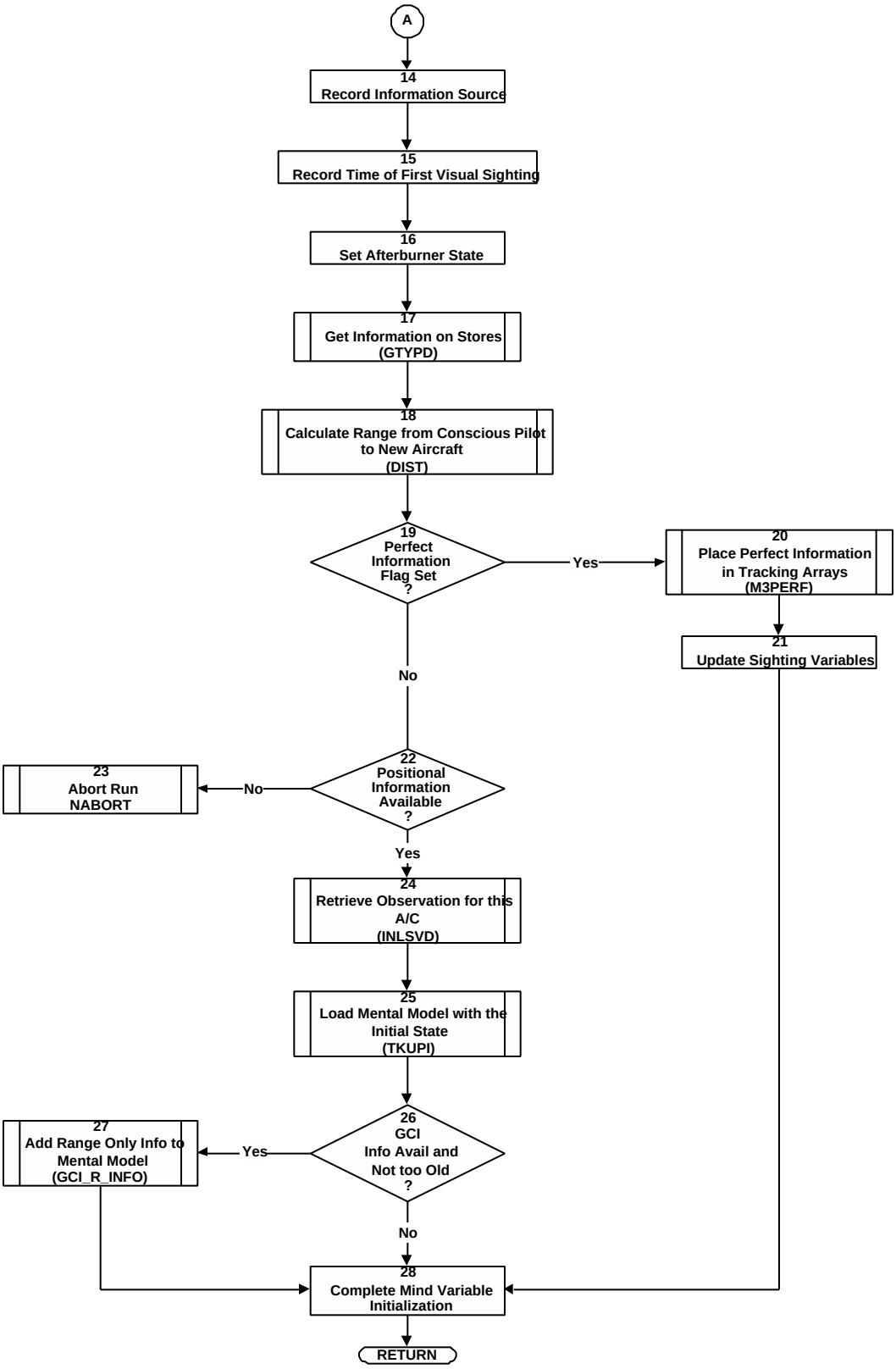

FIGURE 2.15-15. CC2X3 Functional Flow Diagram (Page 2 of 2).

Subroutine MREMAC

Subroutine *mremac* deletes an aircraft from the current mental model. Figure 2.15-16 is the functional flow diagram that describes the logic used to implement *mremac*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Set flag to call a major mental model update.

Block 2. Eliminate acquired missile considerations. If the aircraft being deleted is on the pilot's list of aircraft under attack by command guided missiles that have acquired their targets, remove it.

Block 3. Eliminate missile support considerations. If the aircraft being deleted is on the pilot's list of aircraft under attack by missiles requiring support (semi-active missiles or command guided missiles that have not yet acquired their targets), remove it.

Block 4. Subroutine *anytrk* is called to return any tracks that the conscious pilot has on the aircraft being removed.

Block 5. Test if the conscious pilot has an STT radar track on the aircraft. If true, go to Block 6. Otherwise, jump to Block 7.

Block 6. Subroutine *asklev* is called to force a new decision at the pilot posture level, since the target that the conscious pilot had an STT lock on is now being deleted.

Block 7. Place the aircraft on the dead aircraft list. This involves adding the aircraft's ID to the list of dead aircraft and recording the time that it was observed as dead and its relationship to the conscious pilot.

Block 8. Test if the dead aircraft was friendly or hostile. If hostile, go to Block 9. If friendly, go to Block 10.

Block 9. Since a hostile was just killed, increase the conscious pilot's aggressiveness factor and the overall scale multiplier for offensive values by 25%.

Block 10. Since a friendly was just killed, the aggressiveness factor and offensive multiplier might be reduced to reflect the loss of aggressiveness when a flight member dies. Currently no adjustments are made for this case.

Block 11. Subroutine *minfer* is called to plant a consciousness event to allow for an inferred detection of the hostile that just killed this friendly.

Block 12. Test if the dead aircraft was the conscious pilot's formation (flight or element) leader. If true, promote another pilot (possibly the conscious pilot) in the flight to be flight leader (Blocks 13-25). If not, jump to Block 26.

Blocks 13-25 are executed to determine promotions if the aircraft killed was the conscious pilot's formation leader. There are three possible ranks in the command hierarchy; wingman, element leader and flight leader. The lowest indexed aircraft in an element is the element leader and the lowest indexed element leader is the flight leader, making him the lowest indexed live aircraft in the flight.

Block 13. Test if the conscious pilot is a wingman. For both paths, true or false, continue to determine who should be the flight leader. At this point, if the conscious pilot is not a wingman, he must be an element leader (there are only three assignments possible: wingman, element leader, flight leader, and if he were already the flight leader, we wouldn't be here). For an element leader, go to Block 14. For a wingman, go to Block 21.

Block 14. Find the lowest indexed aircraft in the flight, the lowest indexed element in the flight and the lowest indexed aircraft in the conscious pilot's element. This is done by looping over the aircraft in the conscious pilot's mental model and comparing indices with the current lowest indexed aircraft, replacing the current lowest index if the compared index is lower.

Block 15. Test if the conscious pilot has the lowest index in his element. If true, he should be promoted to at least element leader (Blocks 16-19). If false, the lowest indexed aircraft in the element is promoted to element leader (Block 20).

Block 16. Test if the conscious pilot's element is the lowest indexed element in the flight. If true, the conscious pilot is the lowest indexed aircraft in the lowest indexed element and should be promoted to flight leader (Block 17). If false, the conscious pilot is still the lowest indexed aircraft in his element and should be promoted to element leader, with the lowest indexed aircraft in the flight being promoted to flight leader (Blocks 18-19).

Block 17. The conscious pilot is promoted to flight leader. He is the lowest indexed aircraft in the lowest indexed element. Continue at Block 26.

Block 18. The conscious pilot is promoted to element leader. He is the lowest indexed aircraft in his element.

Block 19. The lowest indexed aircraft in the flight is promoted to flight leader. Continue at Block 26.

Block 20. The lowest indexed aircraft in the element is promoted to element leader. Continue at Block 26.

Block 21. Test if the conscious pilot is an element leader. This is a consistency check. If we reach this block, the conscious pilot must be an element leader (since the flight leader is in the dead aircraft and the conscious pilot is not a wingman, he must be an element leader). If false, go to Block 22. Otherwise, go to Block 23.

Block 22. Subroutine *nabort* is called to terminate the run and print diagnostic messages because the aircraft's role is not flight leader, element leader, or wingman.

Block 23. Test if the conscious pilot is the lowest indexed aircraft in the flight. If true, the conscious pilot is promoted to flight leader (Block 25). If false, the lowest indexed aircraft in the flight is promoted to flight leader (Block 24).

Block 24. The lowest indexed aircraft in the flight is promoted to flight leader.

Block 25. The conscious pilot is promoted to flight leader.

Blocks 26-36 remove the dead aircraft from the conscious pilot's target lists, if it was on them. Memory variables are also cleared of the knowledge of the dead aircraft.

Block 26. Subroutine *asstgt* is called to check the target assignment given to the conscious pilot by his flight leader.

Block 27. Test if the dead aircraft was the conscious pilot's assigned target. If true, go to Block 28. If not, go to Block 29.

Block 28. The orders changed flag is set to true for the conscious pilot. This will force a major update and cause the pilot to reconsider his weapon and target decisions.

Block 29. Test if the dead aircraft was the conscious pilot's selected target. If true, go to Block 30. If not, go to Block 32.

Block 30. Subroutine *gfcsta* is called to retrieve the fire control status for the conscious pilot.

Block 31. The fire control variables for selected missile, selected target id, best target index and missile mode are set to zero for the conscious pilot. These values will be reset when the conscious pilot selects another target in a future consciousness event.

Block 32. If other friendlies had communicated an intent to fire on the dead aircraft, clear this information from the pilot's mental model.

Block 33. If the dead aircraft was not the last aircraft in the mental model, then swap the mental models of the dead aircraft with that of the last one in the mental model. By doing this we effectively remove the dead aircraft from the /*mind3*/ common block arrays by decrementing the number of aircraft in the mental model.

Block 34. Reset values in the /*mind4*/ common blocks that pertain to the dead aircraft.

Block 35. Reset the number of aircraft in the mental model and the number of detected aircraft variables to account for the dead aircraft being removed.

Block 36. Subroutine *recfel* is called to reorder the flight and element lists for the conscious pilot's flight and element in case the dead aircraft was in his element and/or flight.

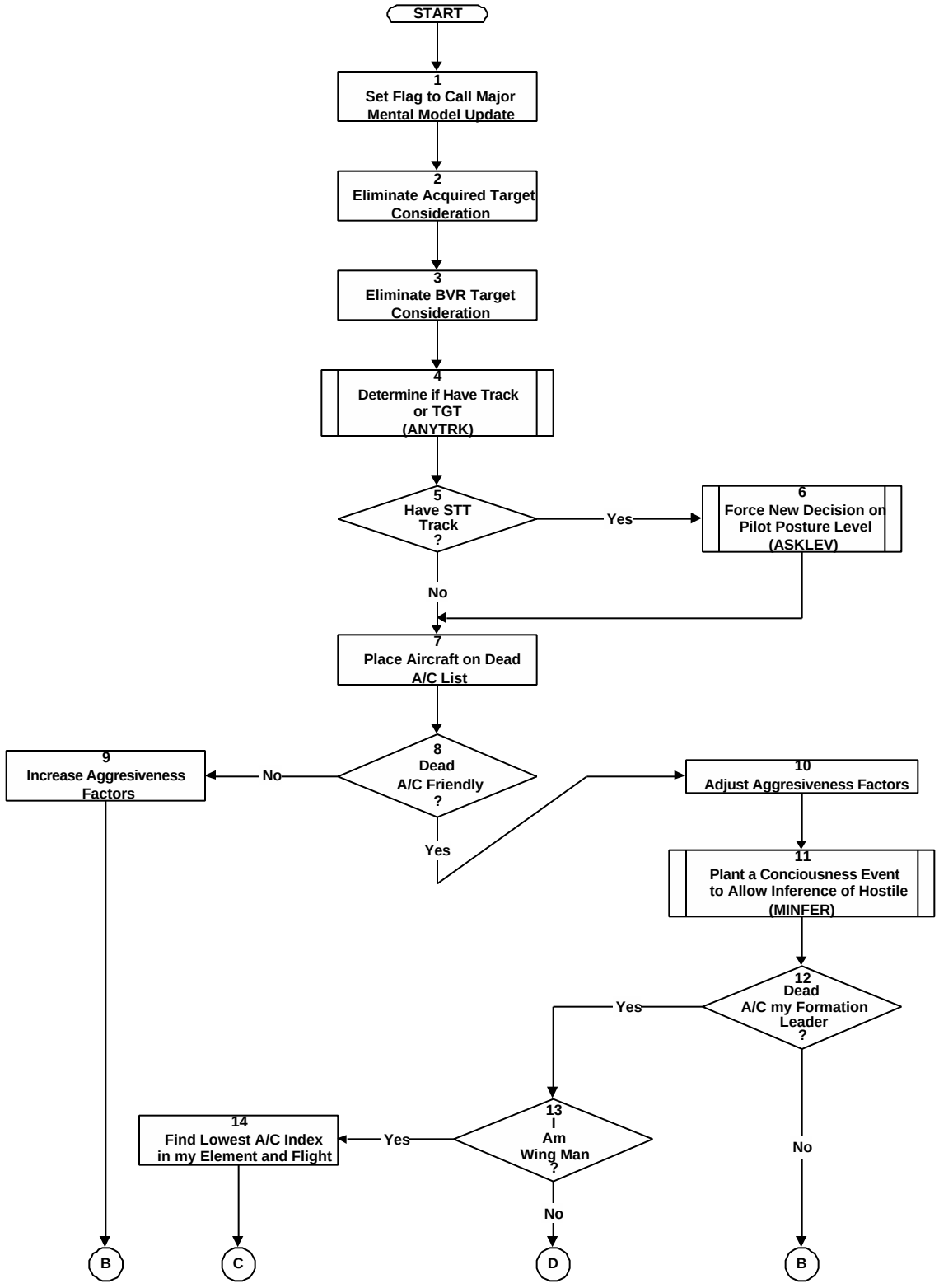

FIGURE 2.15-16. MREMAC Functional Flow Diagram (Page 1 of 3).

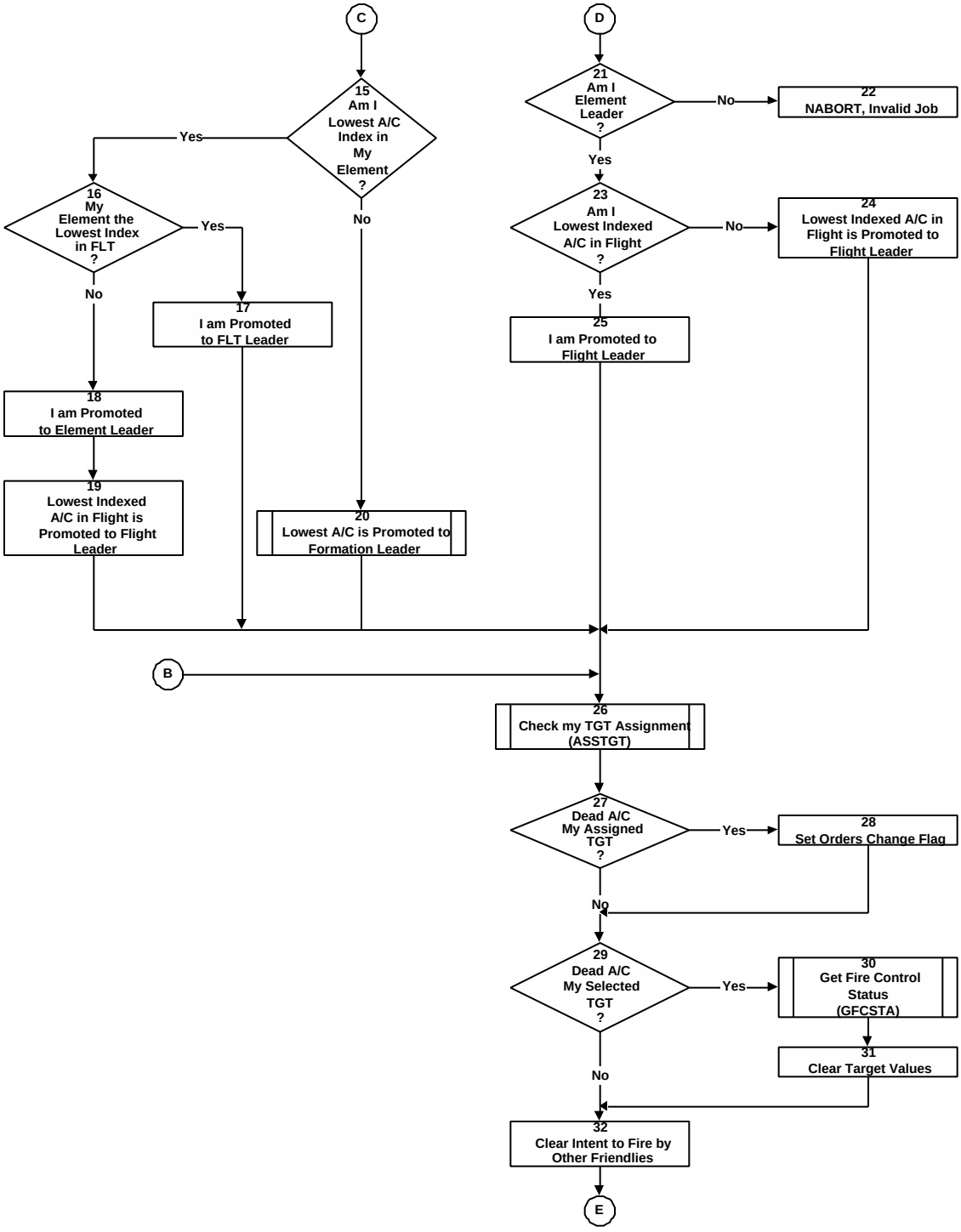

FIGURE 2.15-16. MREMAC Functional Flow Diagram (Page 2 of 3).

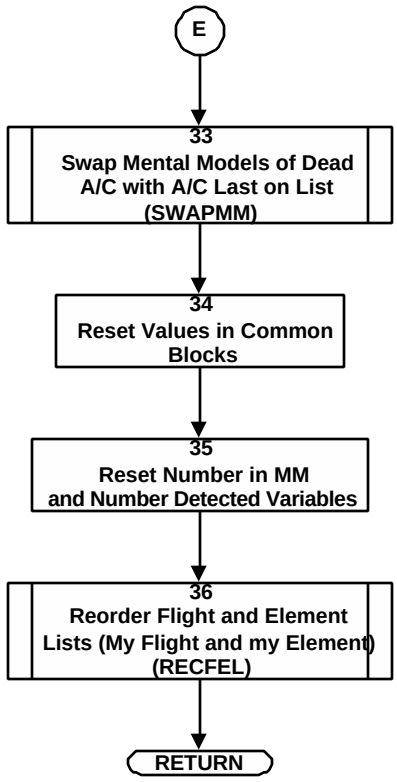

FIGURE 2.15-16. MREMAC Functional Flow Diagram (Page 3 of 3).

Subroutine CC2X2

Subroutine *cc2x2* updates the conscious pilot's perception of previously known aircraft with new information contained in an observation made during this consciousness event. Figure 2.15-17 is the functional flow diagram that describes the logic used to implement *cc2x2*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Test if the perfect information switch is set. If set to true, go to Block 2. Otherwise, jump to Block 3.

Block 2. The time of the observation is reset to the current time when using perfect information. When running in perfect information mode, the mental model track will be updated to the current time with ground truth information by subroutine *m3perf*.

Block 3. Subroutine *vecinc* is called to calculate the position of the observed aircraft at the time of the observation.

Block 4. Subroutine *vecinc* is called again to calculate the position of the conscious pilot's aircraft at the time of the observation.

Block 5. Real function *dist* is called to calculate the range between the conscious pilot's aircraft and the observed aircraft.

In the following blocks, aircraft 'type' refers a type of airframe (e.g. F15, SU27). The 'type quality' refers to how well the target must be identified for firing constraints. This requirement is set by the user. A type quality of *visual_id_md* (the highest, = 4) requires that a visual ID must be made before the pilot may fire. Similarly, a type quality of

c2_id_md (next highest, = 3) requires that a command and control confirmation be issued before the pilot can fire (or a visual ID, since visual is a higher quality, 4 vs. 3).

Block 6. Test if the observation of aircraft type should be accepted. If true, go to Block 7. If not, go to Block 11. The test is a comparison of the type quality of the observation with the type quality of the pilot's current knowledge. If the quality of the observation is higher, use it. If they are the same, then if the observation contains typing information, use it. Otherwise, do not use the typing information in the observation.

Block 7. Save the type quality of this observation.

Block 8. Test if the pilot's knowledge of aircraft type has changed (observation was of equal or better quality and contained actual type information). If the type information is new, go to Block 9. If not, go to Block 18.

Block 9. Set the flag indicating that the observed aircraft's type information has changed.

Block 10. Set the observed aircraft's type in the pilot's mind to the aircraft type from the new observation.

Block 11. Test if the observed aircraft is on the same side as the conscious pilot. If true, establish the relationship between the conscious pilot and the observed aircraft (Blocks 12-13). Otherwise set the relationship to hostile or unknown (Blocks 14-17). Valid relationships are:

1. unknown
2. member of my element
3. member of my flight
4. friendly
5. hostile, not engaged with conscious pilot
6. hostile, engaged with conscious pilot

Block 12. Set the relationship established flag to true.

Block 13. Subroutine *getrel* is called to establish the relationship (see Block 11 for valid relationships).

Block 14. For hostiles, test if the type quality is greater than the required identification mode. If true, set relationship to hostile, not engaged with self (Blocks 15-16). If false, set to unknown (Block 17). Identification modes are:

1.	bvr_id_md	can shoot at unknowns
2.	electronic_id_md	can shoot with any ID method
3.	c2_id_md	can shoot with command and control confirmation
4.	visual_id_md	can only shoot with visual identification

The purpose of this is that the pilot will not definitively flag another aircraft as hostile (and therefore open to attack) until he has a high enough quality identification to satisfy the rules of engagement for this scenario.

Block 15. Set the relationship established flag to true.

Block 16. Set relationship between the conscious pilot and the observed aircraft to hostile, not engaged with self.

Block 17. Set relationship between conscious pilot and observed aircraft to unknown.

Block 18. Test if the observed aircraft was newly typed and is in the detailed consideration group. If true, request a major mental model update for the conscious pilot (Blocks 19-21).

Block 19. Logical function *irhas* is called to determine if the conscious pilot perceives that the observed aircraft has IR missile capability.

Block 20. Set the 'significant information' flag equal to true, indicating that a significant change has occurred in the pilot's situation perception due to this observation. This will trigger a message to be sent by the conscious pilot and place the observed aircraft on the altered aircraft list (see subroutine *cc2x0*).

Block 21. Request a major mental model update for the conscious pilot.

Block 22. If this observation represents a better source of information than the current best source, then update the variable that stores the best source of information available.

Block 23. Test whether the observation contains positional information on the observed aircraft. If true, continue processing the observation data. Otherwise, return control to the calling routine.

Block 24. Test if a visual detection was made. If true, go to Block 25.

Block 25. Set the time of first visual detection if not previously set.

Block 26. Test if the observed aircraft's afterburner state can be determined. If true, go to Block 27. Otherwise, skip to Block 28.

Block 27. Set the observed afterburner state in the mental model of the conscious pilot (on, off or unknown).

Block 28. Test if stores information can be discerned from the observation. If true, go to Block 29. If not, go to Block 32.

Block 29. Test if stores can be identified. If yes, go to Block 30. If no, go to Block 31.

Block 30. Set in the stores identification flag for this aircraft to 2, indicating that the pilot can identify the type and number of stores on the aircraft.

Block 31. Set the stores identification flag for this aircraft to the larger of 1 and it's current value. A value of 1 indicates that stores could be seen but not identified.

Block 32. Subroutine *tkupmm* is called to update the conscious pilot's mental model regarding the physical state of the detected aircraft. *Tkupmm* calls *tkup*, which takes a state vector and an observation and passes them through a Kalman filter to produce an updated state vector.

Block 33. Test if the conscious pilot estimates that the detected aircraft's acceleration is greater than 1G. If true, go to Block 34. If not, go to Block 35.

Block 34. The significant information flag is set to true. This cause the observed aircraft to be added to the altered aircraft list (see subroutine *cc2x0*). Entries on the altered aircraft list will help determine if a major situational update is required.

Block 35. The observation times are reset. The generic observation time *obstim* is set to the detection time. The visual (*seet*) and radar (*radart*) observation times are set if the detection was made by visual or radar means respectively.

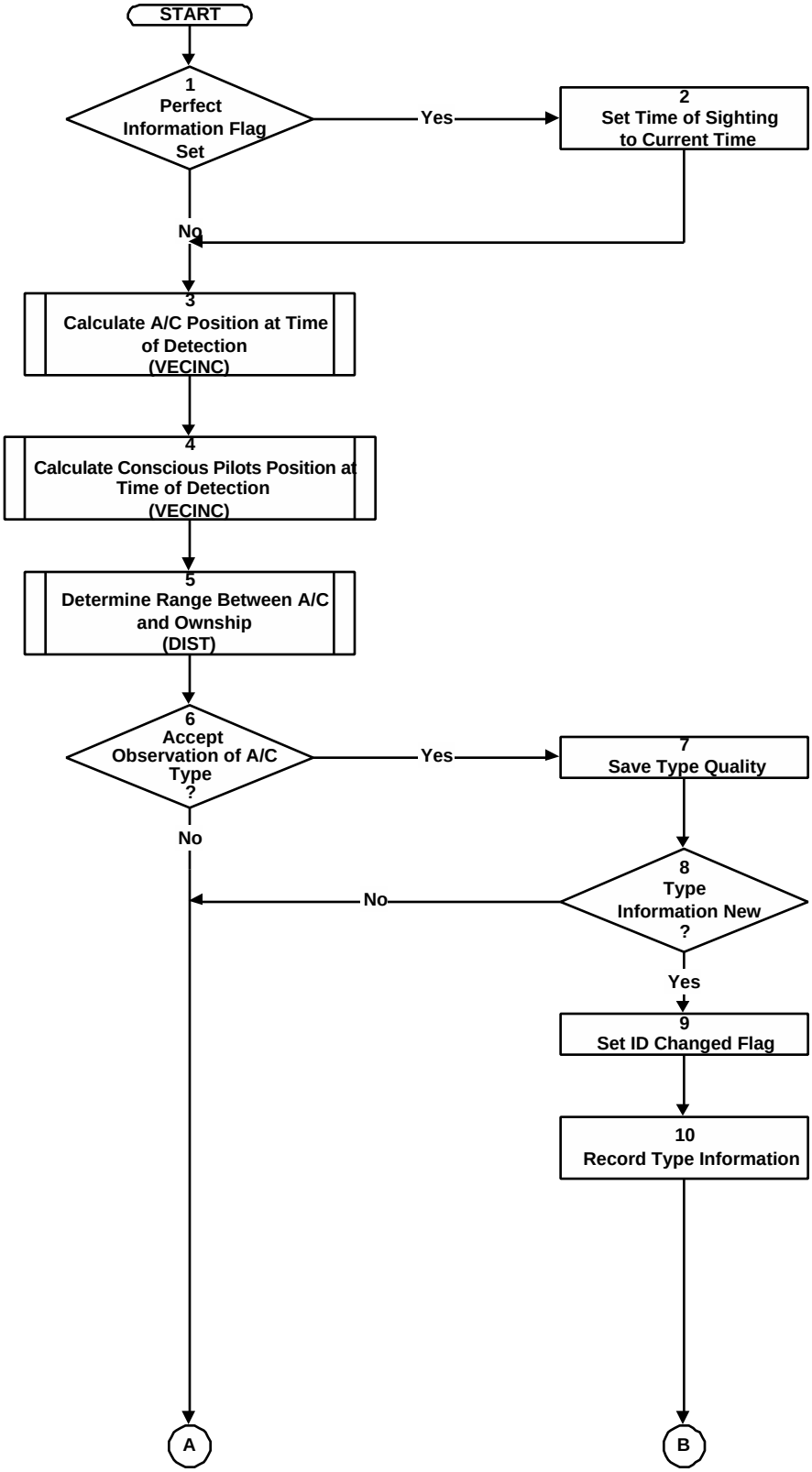

FIGURE 2.15-17. CC2X2 Functional Flow Diagram (Page 1 of 4).

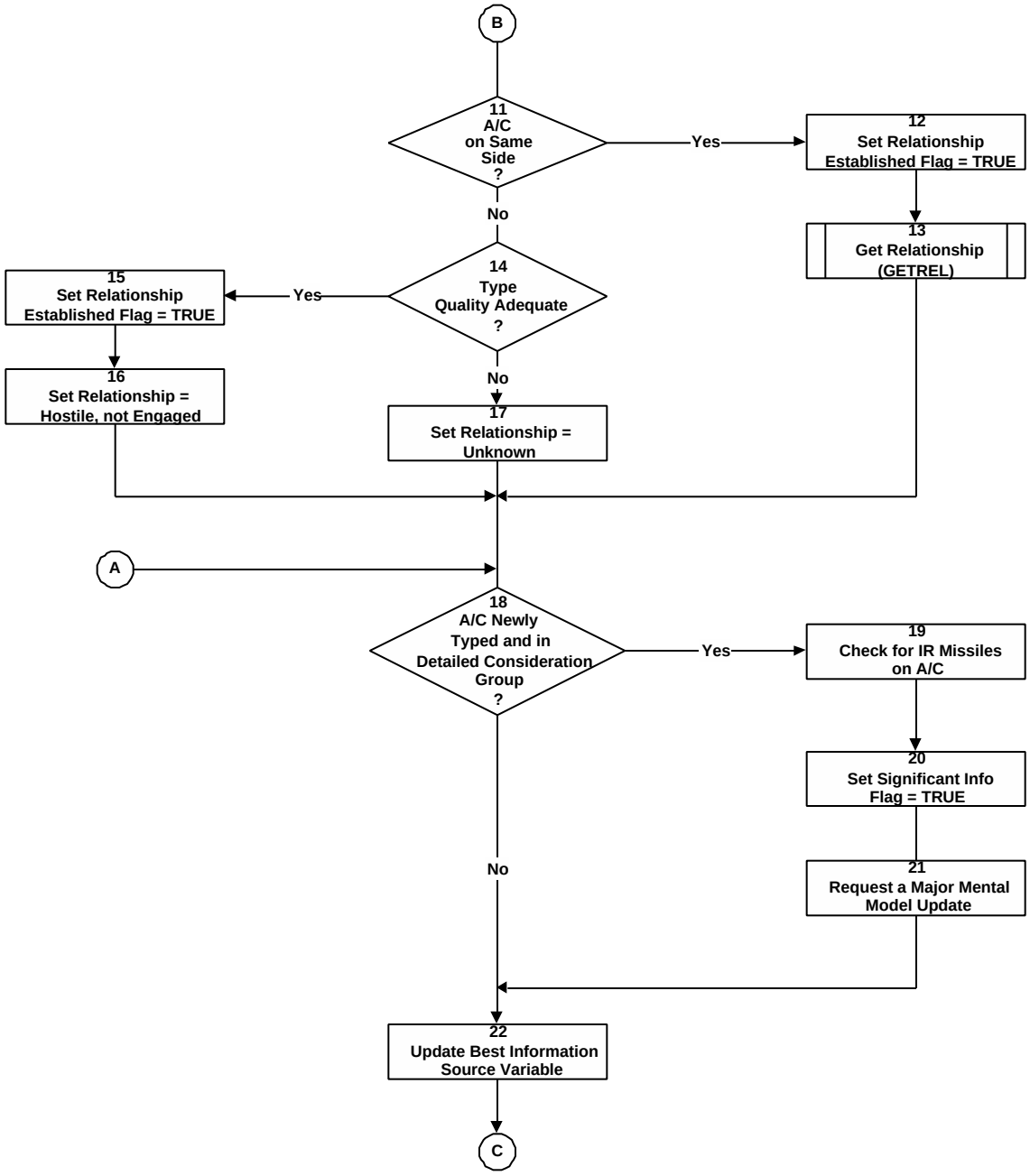

FIGURE 2.15-17. CC2X2 Functional Flow Diagram (Page 2 of 4).

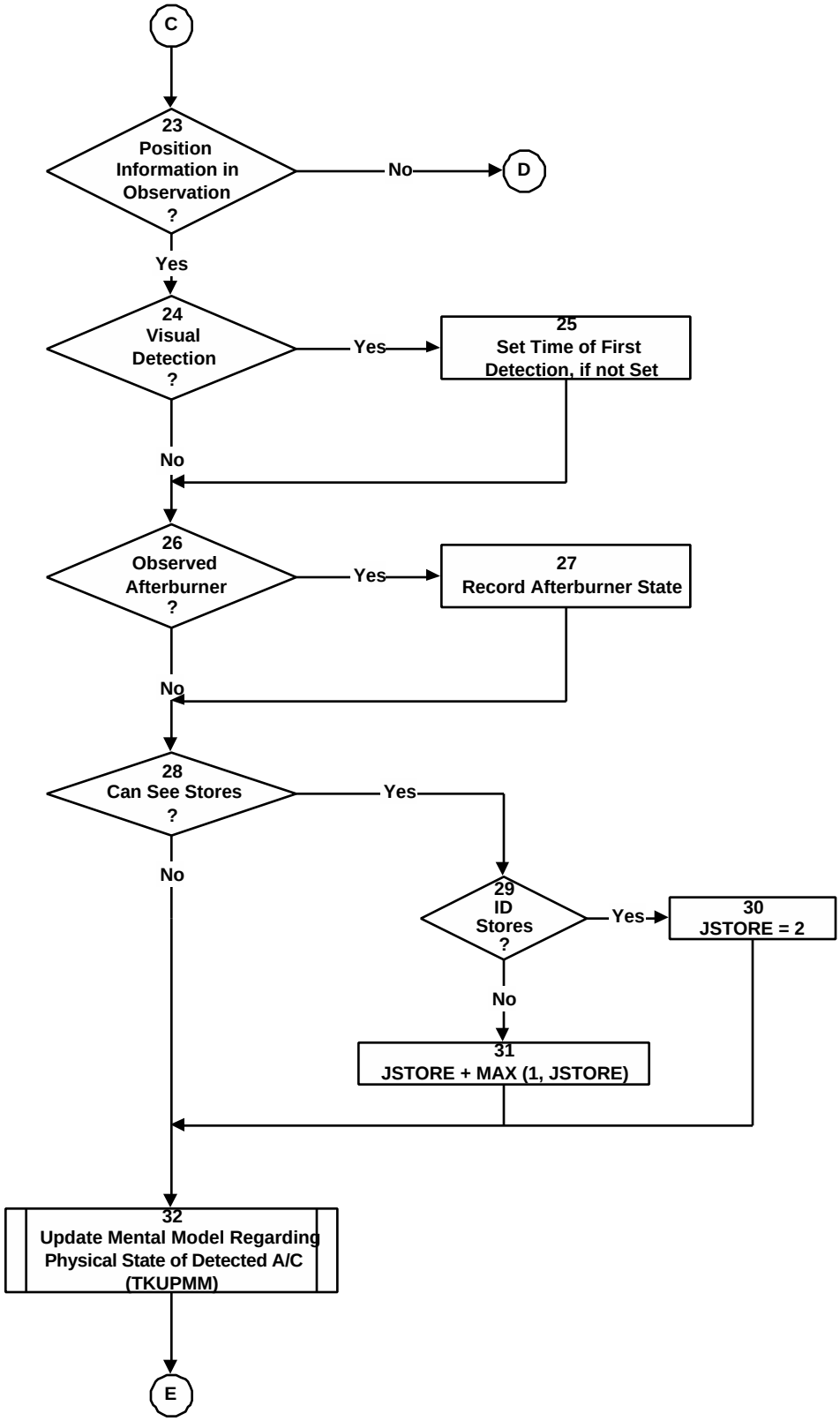

FIGURE 2.15-17. CC2X2 Functional Flow Diagram (Page 3 of 4).

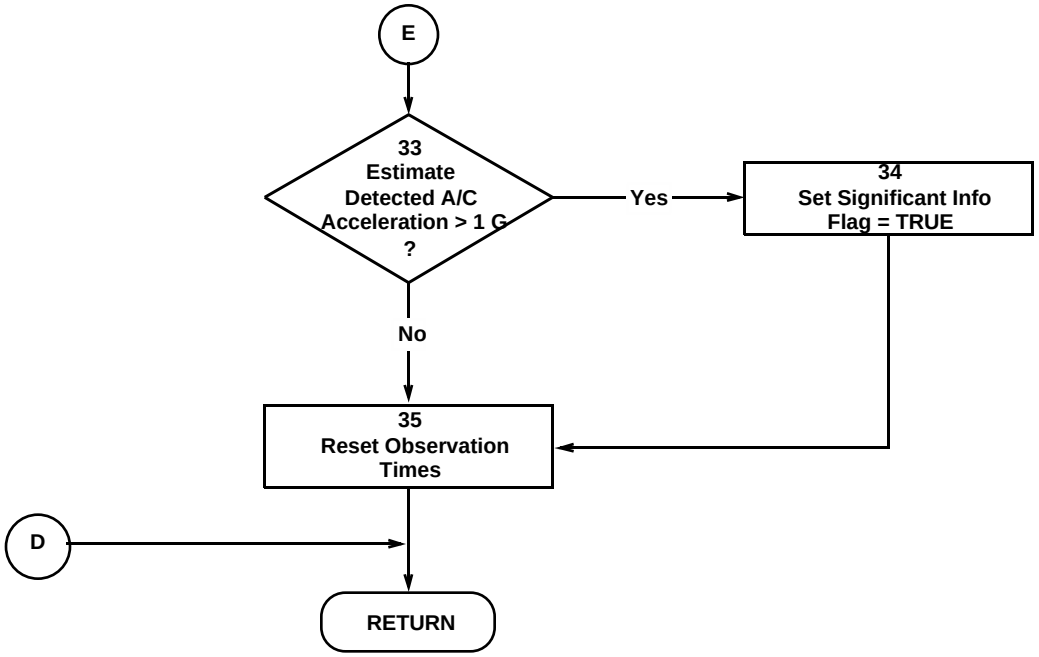

FIGURE 2.15-17. CC2X2 Functional Flow Diagram (Page 4 of 4).

Subroutine ASTYP

Subroutine *astyp* assesses the type of the observed aircraft based on knowledge of the type of other members of its flight. Figure 2.15-18 is the functional flow diagram that describes the logic used to implement *astyp*. The blocks are numbered for ease of reference in the following discussion.

Block 1. Initialize the array that indicates which flights contain typed aircraft.

Blocks 2-4 are in a loop over the aircraft in the conscious pilot's mental model for the purpose of determining which flights contain typed aircraft.

Block 2. Test if the type is known for this aircraft. If true, go to Block 3. If not, go to Block 4.

Block 3. Set the flag to indicate that this aircraft's flight has a typed aircraft in it. This information will be used later to infer a relationship between untyped aircraft and typed aircraft in the same flight.

Block 4. Test if there are more aircraft in the conscious pilot's mental model that still need to be processed. If true, increment the loop counter and proceed to the top of the loop to process the next aircraft (Block 2).

Blocks 5-19 are in a loop over the aircraft in the conscious pilot's mental model for the purpose of finding an aircraft whose type is unknown and is in a flight that contains typed aircraft. An inference will be tested between this aircraft and the typed aircraft in its flight.

Block 5. The flight is determined for the aircraft under consideration.

Block 6. Test if the aircraft is in a flight with no typed aircraft or if the relationship conscious pilot already knows the type of the aircraft under consideration. If true, no inference can be made or the type is already known, so increment the loop counter and go back to the top of the loop to process the next aircraft. If false, then an inference may be drawn, so continue to Block 7.

Blocks 7-18 are in a loop over the aircraft in unknown aircraft's flight. If possible, an inference will be made between this aircraft and the typed aircraft in its flight.

Block 7. Check the relationship of the aircraft selected in the outer loop (unknown to conscious pilot and in flight with known aircraft) to the aircraft with the current (inner) loop index. If the inner loop aircraft is not in the conscious pilot's mental model or is unknown to the conscious pilot, then loop to the next aircraft. Otherwise, go on to Block 8.

Block 8. At this point we have found an unknown aircraft (outer loop) and an aircraft in the same flight that is known (inner loop). Real function *dist* is called twice to calculate the range difference and velocity difference between these two aircraft.

Block 9. A formation factor is calculated using the range difference and the velocity difference between the two aircraft.

Block 10. Test if the formation factor (calculated in Block 9) is less than or equal to the predefined formation limit. If true, the velocity and position differences between the two aircraft are small enough to consider them to be flying in formation. The type for the unknown aircraft may now be inferred to be the same as the known aircraft (Blocks 11-16). If not, jump to Block 18.

Block 11. Set the type and type quality of the unknown aircraft to be the same as for the known aircraft.

Block 12. Set the flag to request a major mental model update (due to newly typed aircraft) for the conscious pilot.

Block 13. Test if the newly typed aircraft is on the same side as the conscious pilot. If true, go to Block 14. If not, go to Block 15.

Block 14. Subroutine *getrel* is called to get the relationship between the conscious pilot and the newly typed aircraft. Continue at Block 19.

Block 15. Test if the type quality for the newly typed aircraft is greater than or equal to the required identification mode. If true, go to Block 16. If not, go to Block 17.

Block 16. Set the relationship between the conscious pilot and the newly typed aircraft to hostile, not engaged with self. Continue at Block 19.

Block 17. Set the relationship between the conscious pilot and the newly typed aircraft to unknown. Continue at Block 19.

Block 18. Bottom of the inner loop. Test if there are any more aircraft in the flight that can be checked. If true, increment the loop counter and proceed to the top of the inner loop to process the next aircraft. If not, go to Block 19.

Block 19. Bottom of the outer loop. Test if there are any more aircraft in the conscious pilot's mental model that need processing. If true, increment the loop counter and proceed to the top of the outer loop to process the next aircraft. If not, return.

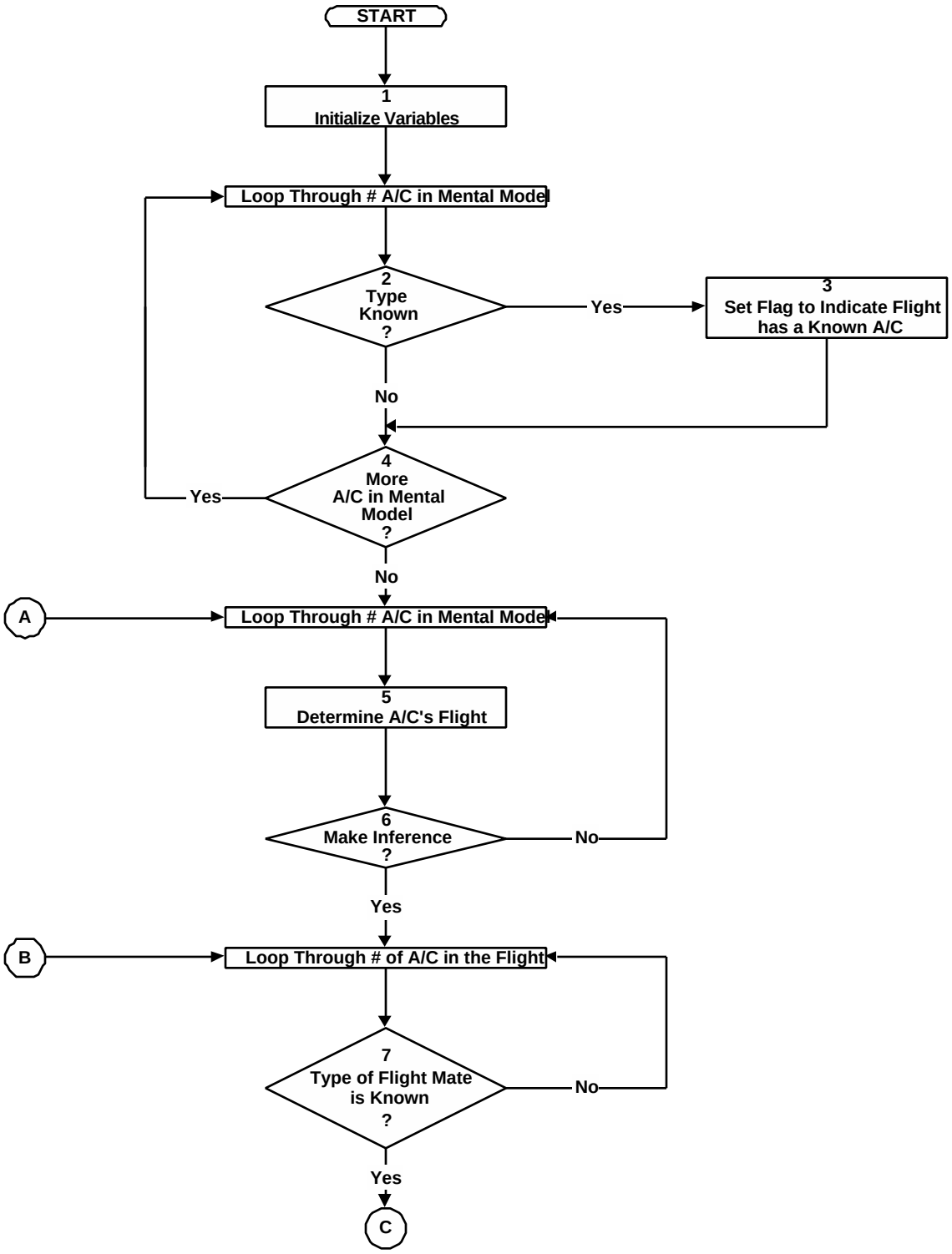

FIGURE 2.15-18. ASTYP Functional Flow Diagram (Page 1 of 2).

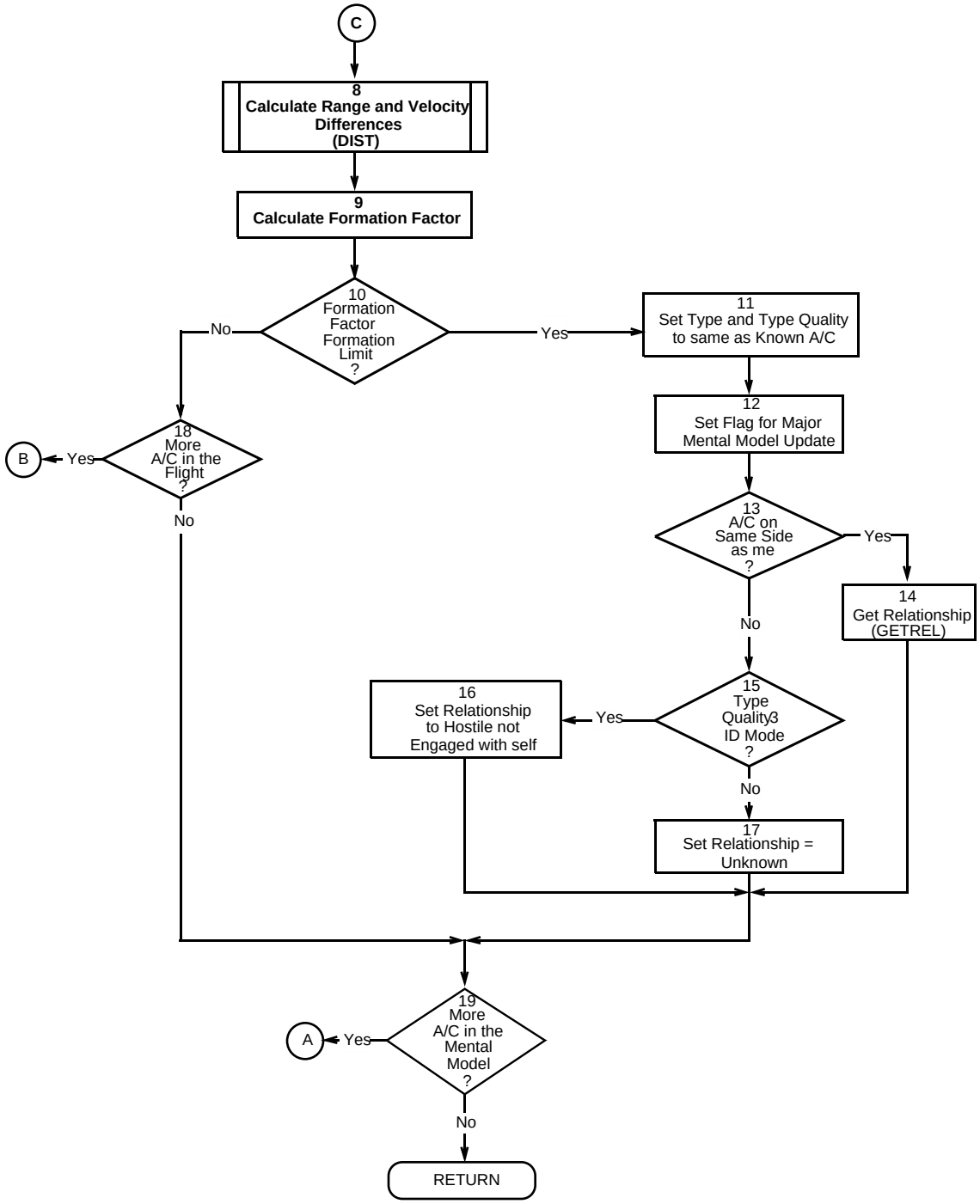

FIGURE 2.15-18. ASTYP Functional Flow Diagram (Page 2 of 2).

Subroutine CC2X4

Subroutine *cc2x4* updates all of the mental model tracks of the conscious pilot to the current time. Figure 2.15-19 is the functional flow diagram that describes the logic used to implement *cc2x4*. The blocks are numbered for ease of reference in the following discussion.

Block 1. This block represents the start of a manual do loop, looping over the number of tracks. The loop counter is incremented each time this block is entered. A manual do loop is used here because aircraft that have not been observed for a long period of time are removed from the mental model, which would have the effect of incrementing the aircraft index currently under consideration if using a conventional do loop. The aircraft index should not be incremented for discarded aircraft.

Block 2. Test if the loop counter is greater than the number of tracks in the mental model. If true, the loop is finished and the subroutine is exited (Block 3).

Block 3. This is the normal exit point for this subroutine. Control is returned to the calling routine.

Blocks 4-7 are in a loop over the number of aircraft detected.

Block 4. Test if the detection being examined was for the aircraft under consideration. If true, check to see if this detection contained positional information (Blocks 5-6). Otherwise, proceed to the next detected aircraft (Block 7).

Block 5. Tese if this detection contained positional information. If true, go to Block 6. If not, go to Block 7.

Block 6. Set the 'aircraft was seen' flag to true. This will be used later (Blocks 8 and 23) to determine how the track should be propagated.

Block 7. Test for more observations to be examined. If true, increment the counter go back to the top of the inner loop (Block 4).

Block 8. Test if the 'aircraft was seen' flag is true. If true, jump to Block 21. If not, go to Block 9.

Block 9. De-type an aircraft that has not been recently observed (logical function *detype* is called to make this determination, basically non flight members that are relatively close to unknowns or aircraft on the opposite side and have not been observed within the last 20 seconds are de-typed). If the aircraft is to be de-typed, go to Block 10. If not, go to Block 12.

Block 10. Set the flag to request a major mental model update.

Block 11. Unset the variables that contain typing information. Variables that record aircraft ID and type are set to zero, the type quality variable is set to its lowest level, and the flag that indicates a change in aircraft ID status is set to true.

Block 12. Test if it has been a long time since the aircraft under consideration has been observed or assessed to be the owner of a missile. If true, go to Block 13. If false, go to Block 14.

Block 13. Subroutine *mremac* is called to remove an aircraft from the conscious pilot's mental model. From here, go back to the top of the outer loop to process the next mental model track (Block 2).

Block 14. Test if the aircraft under consideration can be assessed as dead. If true, go to Block 15. If not, go to Block 17.

Block 15. Subroutine *addmsg* is called to add a message regarding the death of the aircraft under consideration to the list of outgoing messages.

Block 16. Subroutine *mremac* is called to remove an aircraft from the conscious pilot's mental model. Jump back to Block 2.

Block 17. Subroutine *vsub* is called to calculate the difference in position between the conscious pilot's aircraft and the aircraft under consideration.

Block 18. Test if the difference in position between the conscious pilot's aircraft and the aircraft under consideration is too large for a pseudo track. If true, set the 'too_far' flag to true and continue at Block 23. Otherwise, determine if the uncertainty in position is too great for a pseudo track (Blocks 17-20).

Block 19. Subroutine *mudpack* is called to fill the cross correlation matrix *ldxcorr*, that will be used in determining if the position uncertainty is large for the aircraft under consideration.

Block 20. Subroutine *trk_vars_x* is called to form variances from positional cross correlation.

Block 21. Test if the positional variance is too large to use a pseudo observation. If the variance is too large, go to Block 22. Otherwise, go to Block 23.

Block 22. Set the 'not_good' flag equal to true, indicating that the positional uncertainties of the aircraft under consideration are too large to attempt to update the track using the pseudo observation method.

Block 23. Test if the mental model state vector time is equal to the current time. If true, go to Block 24. If not, go to Block 25.

Block 24. The track is already up to date. Set the track update method to 'no time', update some bookkeeping variables, increment the main loop counter and then proceed to the top of the loop to process the next aircraft (Block 2).

Block 25. Decision whether to use a Kalman filter to propagate the track. If one of the following four conditions is met, use a Kalman filter to propagate the track:

1. Aircraft under consideration 'was seen' (detection with positional information)
2. The *pmimpt* mode flag was set to true (indicating use a Kalman filter, false means use pseudo observation technique)
3. The aircraft is too far away for a pseudo track
4. The range uncertainty between the ownship and the aircraft under consideration is too great for a pseudo track

Blocks 26-28 are executed if a Kalman filter is chosen as the method for track propagation.

Block 26. Reduce the perceived accelerations of the aircraft under consideration.

Block 27. Reduce the perceived vertical velocity of the aircraft under consideration.

Block 28. Subroutine *kalmni* is called to propagate the track using a Kalman filter. Continue at Block 35.

Block 29. Test if the aircraft under consideration is outside the detailed consideration group. If true, this is a low priority target. Go to Block 30. If not, continue at Block 31.

Block 30. Subroutine *vecinc* is called to propagate the track using a constant velocity projection. Go to Block 35.

Block 31. Test if the perfect information flag is set. If true, go to Block 32. Otherwise, go to Block 34.

Block 32. Subroutine *m3perf* is called to use ground truth information to update the pilot's perception of the target.

Block 33. Remove the aircraft under consideration from conscious pilot's mental model if its altitude is less than or equal to zero (subroutine *mremac*), indicating that it has hit the ground.

Block 34. Subroutine *msvpob* is called to make a track update using the pseudo observation technique.

Block 35. Update the mental model state vector time to be the current time.

Block 36. Update current speed and time lagged velocity for this track. Go back to Block 2.

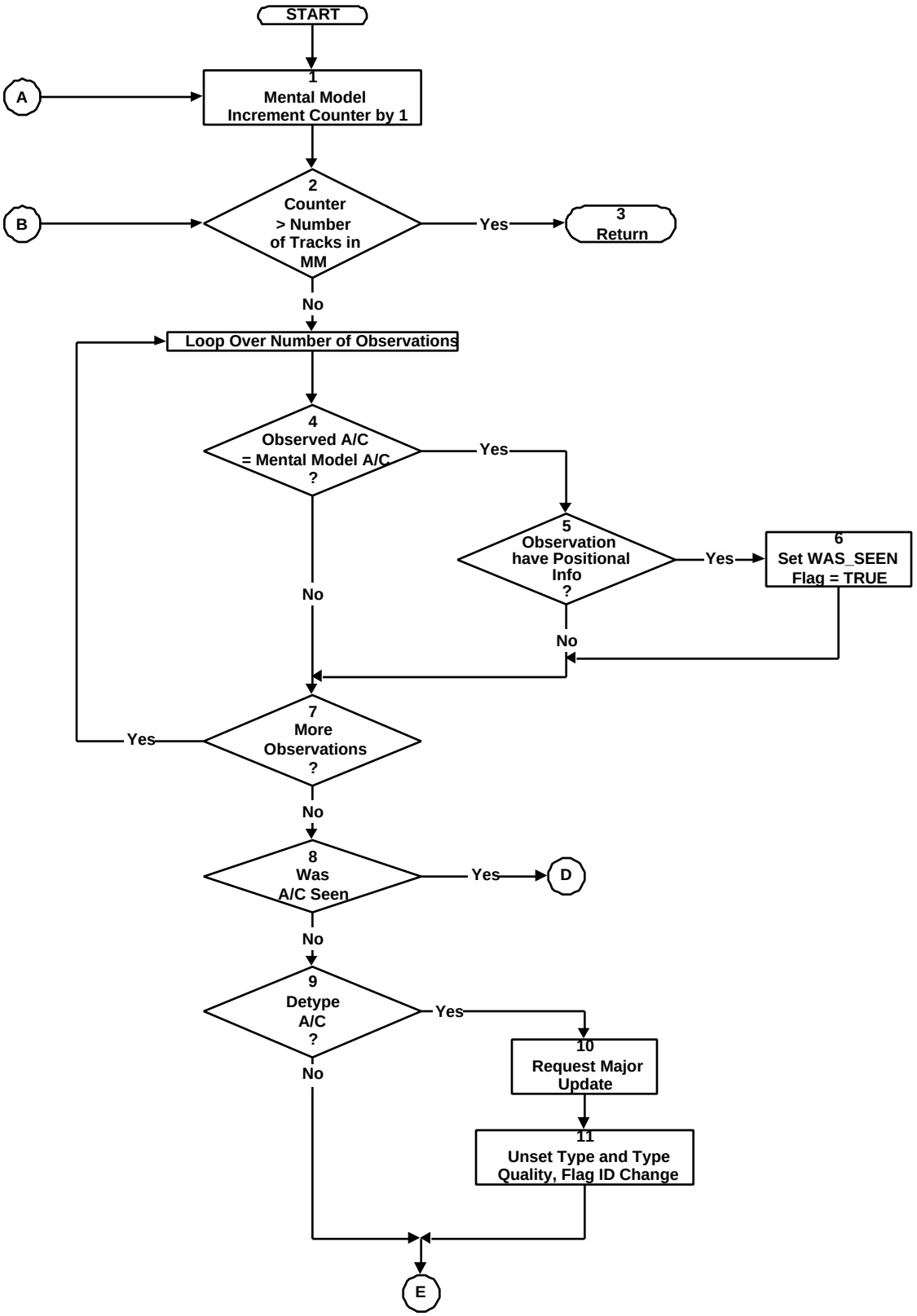

FIGURE 2.15-19. CC2X4 Functional Flow Diagram (Page 1 of 3).

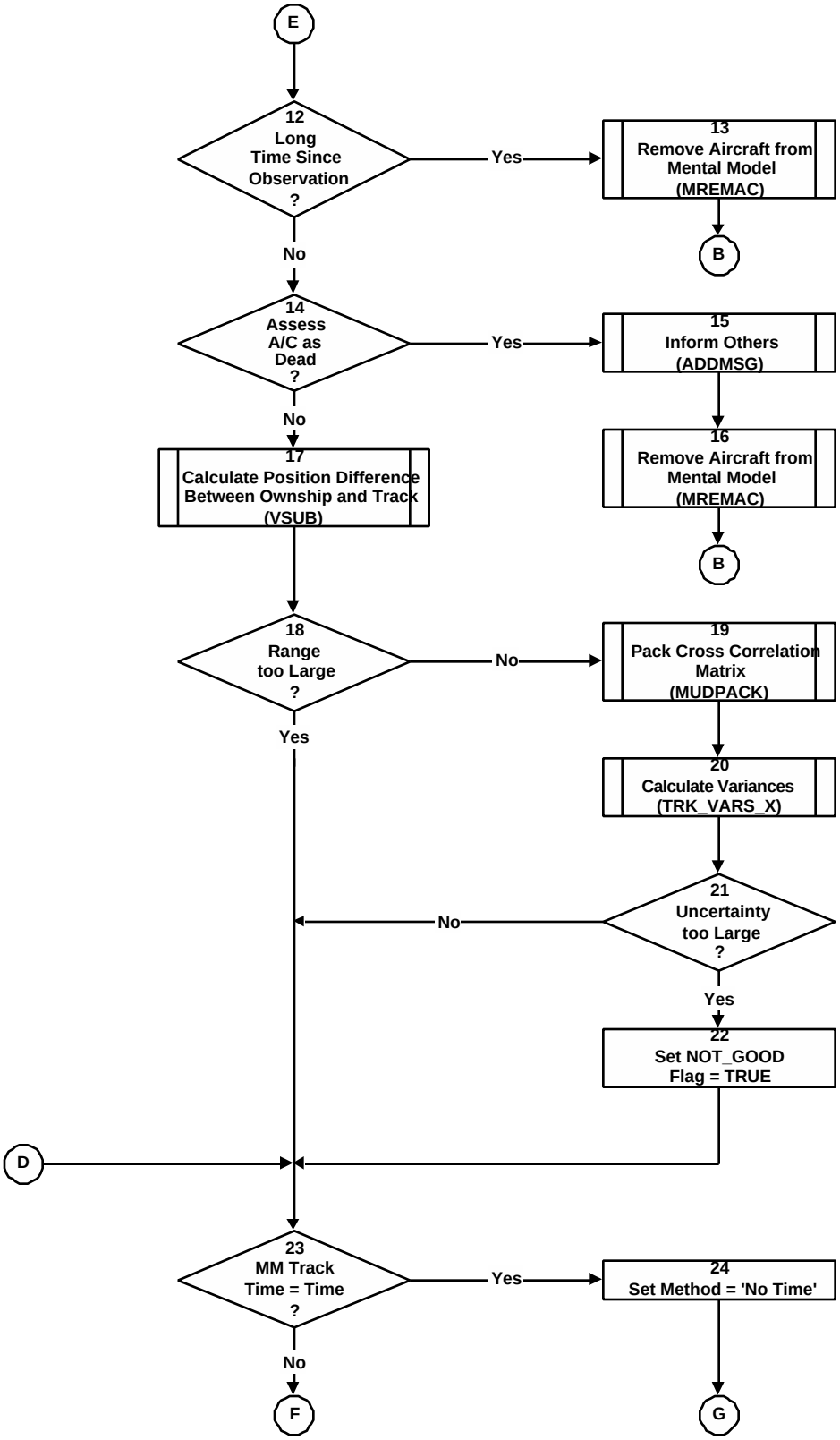

FIGURE 2.15-19. CC2X4 Functional Flow Diagram (Page 2 of 3).

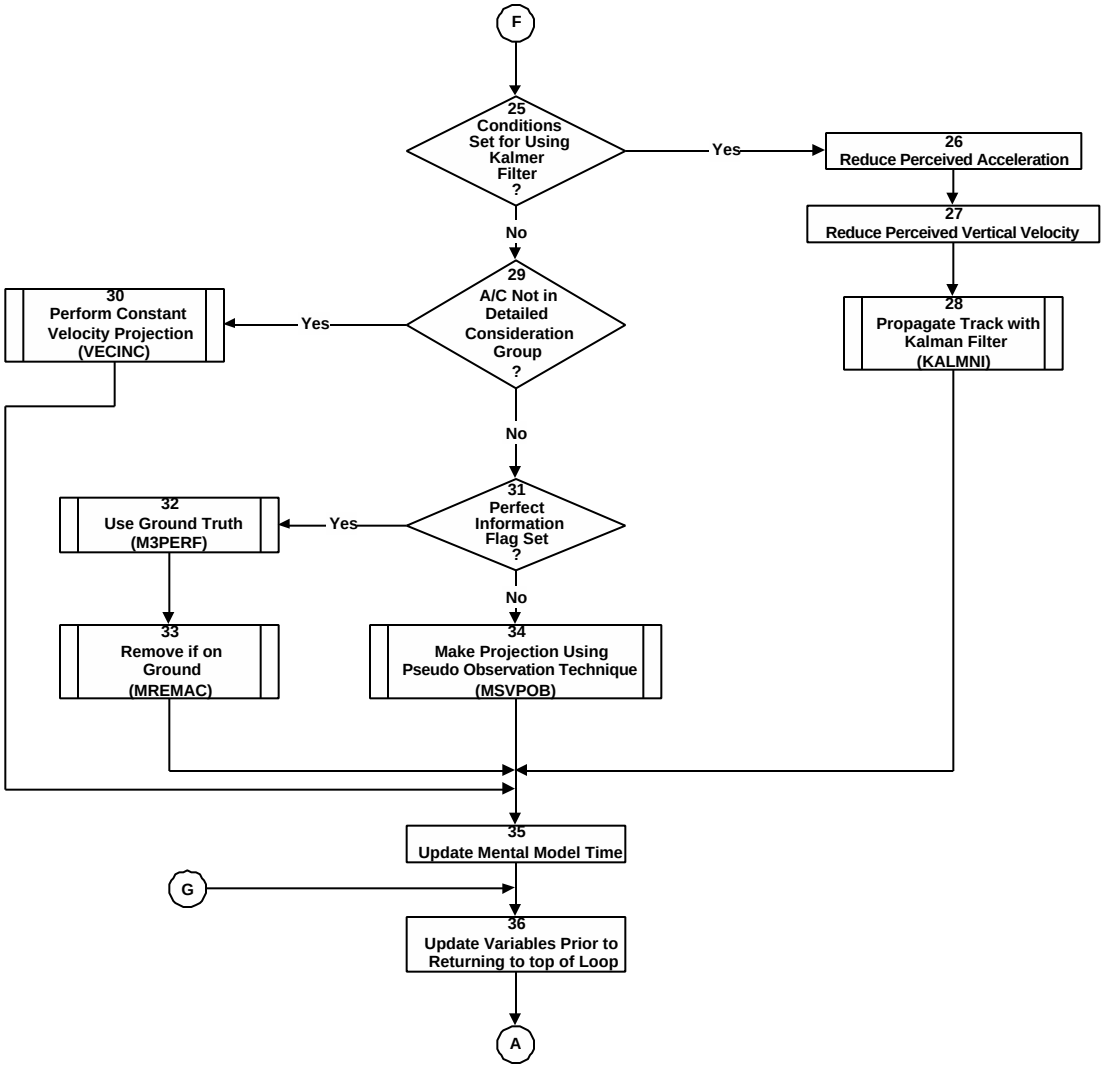

FIGURE 2.15-19. CC2X4 Functional Flow Diagram (Page 3 of 3).

2.15.4 Assumptions and Limitations

If a detection is made of a platform that has a mach number of zero, the platform is presumed to be a SAM.

SAM sites are restricted from making visual observations, only the SAM sensors can provide observations to the SAM.

Aircraft are allowed a visual observation of an obstructed visual sector (down and behind) for the first pass only (to allow observation of flight mates).

Inferred detection of SAM sites is prohibited.

When the sensor routine projects an aircraft's trajectory from the last state vector time to the current time or to the time of observation, the projection is made assuming a constant longitudinal and transverse acceleration.

Pilots have perfect knowledge of their own aircraft's state and performance capabilities.

Update of knowledge of self takes no time.

Correlation of observations with information already existing in the pilot's mind is assumed to be perfect.

No incorrect typing of aircraft.

No false aircraft or missile tracks.

2.15.5 Known Problems or Anomalies

The units are incorrect in the Mach test in Block 11 of subroutine *perfrm*. This has already been corrected for Version V6.3.

The test for significant change in Block 33 of *cc2x2* may cause the flag to be unset if it was previously set but the target aircraft is observed to be pulling less than 1 G. This is unlikely, and the effect would be that the next major mental model update would occur on schedule instead of being done early.